Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Evaluación de indicadores de calidad del sistema de Vigilancia MLAT del Aeropuerto LEBL según MOPS EUROCAE ED-117

TITULACIÓN: Grado en ingeniería de Sistemas Aeroespaciales

AUTOR: Marc Klingenberg Campoy

DIRECTOR: Albert Prades Gimeno

FECHA: 06 de septiembre del 2019

Resumen

El objetivo de este trabajo consiste en crear una aplicación que permita evaluar ciertos parámetros de calidad del sistema de Multilateración del aeropuerto de Barcelona a partir de datos de tráfico de oportunidad que el propio sistema genera.

El conjunto de parámetros que se van a evaluar son los que se especifican en el documento de EUROCAE “Minimum operational performance specification for Mode S Multilateration systems for use in advanced surface movement guidance and control systems” (ED-117). Este documento declara unos valores mínimos de calidad para diferentes aspectos de actuación del sistema MLAT.

La aplicación permite decodificar los ficheros que genera el sistema MLAT y calcular el valor de los parámetros deseados. A través de la interfaz de la aplicación se pueden observar los resultados de la evaluación, así como exportar los datos finales para poder usarlos posteriormente.

Principalmente la aplicación está enfocada para su uso en ENAIRE, dónde existe la necesidad de verificar cada cierto tiempo que se cumple con los estándares para cumplir con auditorías de agencias cómo AESA. Ya existe una aplicación que calcula estos parámetros, aunque en vez de trabajar con tráfico de oportunidad trabaja con datos generados por un vehículo usado expresamente para la evaluación. Este vehículo realiza un recorrido en el aeropuerto mientras se monitorea su posición con un GPS diferencial. El objetivo de esta aplicación es obtener el valor de los parámetros sin la necesidad de usar un vehículo expresamente para ello, y por lo tanto aumentar la frecuencia con la que se hacen evaluaciones del sistema MLAT.

Una parte del trabajo se centra también en lidiar con una serie de vehículos terrestres que aparecen en las grabaciones de tráfico de oportunidad y que no transmiten de una manera nominal, por lo que empeoran los resultados del cálculo de los diferentes parámetros. Es por eso que la aplicación también tiene una interfaz que permite eliminar dichos vehículos de la evaluación.

Title: Evaluation of quality indicators of MLAT Surveillance systems at LEBL Airport according to MOPS EUROCAE ED-117

Author: Marc Klingenberg Campoy

Director: Albert Prades Gimeno

Date: September 06, 2019

Overview

The aim of this project is to develop an application that allows the evaluation of certain performance parameters of the Multilateration system of Barcelona airport, based on opportunity traffic data generated by the system itself.

The set of parameters which are evaluated are specified in the EUROCAE document "Minimum operational performance specification for Mode S Multilateration systems for use in advanced surface movement guidance and control systems" (ED-117). This document states the minimum performance values for different aspects of the MLAT system.

The application decodes the files generated by the MLAT system and calculates the value of the desired parameters. Through the application interface you can see the results of the evaluation, as well as export the final data for later use.

The application is mainly focused to be used in ENAIRE, where there is the need to verify periodically that the standards are met in order to comply with audits from agencies such as AESA. There already exists an application that calculates these parameters, although instead of working with opportunity traffic, it works with data generated by a vehicle used expressly for the evaluation. This vehicle makes a predefined route through the airport while its position is being monitored with a differential GPS. The objective of this application is to obtain the value of the parameters without the need to use a vehicle expressly for it, and therefore increase the frequency of the MLAT system evaluations.

A part of the work also focuses on dealing with a series of ground vehicles that appear in the recordings of opportunity traffic and that do not transmit in a nominal way, so the results of the calculation of the different parameters gets worse. That is why the application also has an interface that allows you to discard these vehicles from the evaluation.

ÍNDICE

INTRODUCCIÓN

La finalidad de este trabajo es evaluar ciertos parámetros de calidad del sistema MLAT de Barcelona. Los parámetros que se van a evaluar son los que se especifican en el documento de EUROCAE “Minimum operational performance specification for Mode S Multilateration systems for use in advanced surface movement guidance and control systems” (ED-117).

Para evaluar estos parámetros se crea una aplicación, que será la parte más importante y producto final del trabajo. Esta aplicación debe ser capaz de decodificar los datos que produce el sistema MLAT, y a través de un algoritmo calcular los diferentes parámetros requeridos, teniendo en cuenta las condiciones que se especifican en el documento ED-117. Una vez calculado el valor de cada parámetro, la aplicación debe presentar de manera clara los resultados y debe permitir exportarlos para poder trabajar con ellos a posteriori.

Los datos que requiere la aplicación son los ficheros que produce la MLAT (.ast). Estos ficheros están compuestos de paquetes que contienen información de posición; velocidad; matricula del aeronave; y otros parámetros de cada uno de los blancos que se encuentra activo en el aeropuerto en un determinado instante de tiempo. Estos blancos son aviones que están usando las instalaciones en el momento de la grabación y el trabajo de referirá a ellos como trafico de oportunidad.

A parte de los aviones, en las grabaciones también aparecen algunos vehículos terrestres que están circulando por el aeropuerto. La aplicación será capaz de descartar la mayoría de estos vehículos de la evaluación, puesto que no transmiten en una tasa nominal y empeoran los resultados de la evaluación.

Durante el trabajo se ha descartado la posibilidad de calcular el parámetro referente al error en posición con solo datos de tráfico de oportunidad, debido principalmente a la falta de información contenida en los ficheros que genera el sistema MLAT de Barcelona. Es por esto que para calcular dicho parámetro si es necesario el uso de un vehículo con un equipo de GPS diferencial a bordo, y por lo tanto la aplicación es capaz de tratar con dicha información. Aún así, se hace un estudio de la posible implementación del cálculo de este parámetro con trafico de oportunidad para futuras actualizaciones previstas en el sistema MLAT, dónde se mejorarán sus capacidades y se podrán obtener datos referentes a este campo.

Se ha descartado también la opción de evaluar de manera análoga al MLAT el sistema SMR de Barcelona y obtener el valor de los parámetros especificados en el documento ED-116 de EUROCAE. Se ha descartado debido a que los ficheros generados por el sistema SMR contienen muy poca información, lo cual dificulta la creación de un algoritmo que trate de manera genérica con

estos datos y calcule los diferentes parámetros, aún así se contempla la inclusión del análisis del SMR como una actualización de la aplicación.

La conclusión del trabajo es que se ha conseguido crear una aplicación completamente funcional que evalúa el funcionamiento del sistema MLAT con tráfico de oportunidad, diferenciándose así de las aplicaciones que ya existían hasta día de hoy, las cuales solo funcionaban con datos complementarios de GPS diferencial. Solo hay que hacer inciso en que no se ha conseguido evaluar el parámetro referente al error en posición con tráfico de oportunidad, dejando así este campo como una posible continuación del trabajo en cuanto se actualice el sistema MLAT de Barcelona.

Respecto al funcionamiento de la MLAT, se determina que funcionamientos es correcto en su cómputo global, con algunas excepciones que podrían ser mejoradas para acabar de cumplir con las especificaciones del MOPS de EUROCAE.

El documento se ha dividido en los siguientes capítulos:

1. **Obtención y adaptación de datos:** Son los datos que necesita la aplicación para poder evaluar el sistema MLAT. Consta de los ficheros de tráfico de oportunidad, así como de mapas del aeropuerto y coordenadas de elementos como el centro de coordenadas que usa la MLAT. Estos datos también hay que adaptarlos, ya sea decodificando el fichero que produce la MLAT o transformando datos de puntos que componen el aeropuerto del sistema WGS84 al plano cartesiano.
2. **Segmentación de LEBL:** Muchos parámetros a evaluar tienen diferentes mínimos según el tipo de área en la que se encuentra la aeronave (taxi, stand, runway…). Es por eso que hay que segmentar el aeropuerto de Barcelona en las diferentes áreas que lo conforman y que se requieren para la evaluación, así cómo crear un algoritmo que clasifique cada punto del fichero de tráfico de oportunidad y le asigne un área dentro de la aplicación.
3. **Parámetros ED-117:** En este capítulo se entra en detalle en cada parámetro y se especifica cómo se calculará su valor en la aplicación.
4. **Vehículos descartados:** Se especifica que tipos de vehículos se descartan de la aplicación y el porqué.
5. **Resultados evaluación MLAT LEBL**: Se muestran los resultados que da la aplicación en la evaluación de un fichero del sistema MLAT, comparando los resultados de tráfico de oportunidad con los obtenidos con datos de D-GPS.

Las palabras y abreviaciones clave para la comprensión del documento son:

ASTERIX: All-purpose structured EUROCONTROL surveillance information exchange.

MLAT: Multilateración.

SMR: Surface movement radar.

D-GPS: GPS diferencial.

ADS-B: Sistema de vigilancia. Automatic dependent surveillance-broadcast system.

LEBL: Código OACI del aeropuerto de Barcelona.

WGS84: Sistema de coordenadas. World Geodetic System 84 (1984).

NED: Sistema de coordenadas. North East Down.

SIC: System Identification Code.

UTC: Coordinated Universal Time.

FL: Flight level.

ATS: Air traffic services.

ATC: Air traffic control.

CAPITULO 1. Obtención y adaptación de datos

1.1 Definición del Sistema MLAT

La multilateración es una técnica usada en navegación y sistemas de vigilancia.

Basa su funcionamiento en estaciones receptoras colocadas en posiciones conocidas. Calcula la posición del emisor midiendo la diferencia de tiempo de llegada de una señal del emisor a un cierto número de estaciones receptoras sincronizadas. Con los datos de dos estaciones se crean un hiperboloide que contiene las posibles posiciones del blanco, con tres estaciones se crean dos hiperboloides que intersecan en una curva, y con cuatro estaciones la intersección de dos curvas forma uno o dos puntos posibles. La precisión de este método depende del número de receptores así de su posición respecto al blanco.

El sistema MLAT codifica sus mensajes de acuerdo a las categorías 10 y 20 de ASTERIX.

1.2 Decodificación de ficheros del sistema MLAT

Las grabaciones que el sistema MLAT lleva a cabo se codifican siguiendo las especificaciones ASTERIX de la organización EUROCONTROL. ASTERIX es un estándar en el método de intercambio de información de ATS. EUROCONROL ha sido su desarrollador y se encarga del mantenimiento y de sus actualizaciones. Está estructurado en categorías y cada una está enfocada a un tipo de información diferente a transmitir. Para poder decodificar estos mensajes EUROCONTROL proporciona una lista de especificaciones, una para cada categoría. Además también proporciona un documento que explica la estructura general de los mensajes.

Las especificaciones para la MLAT se encuentran en las categorías 10 y 20, por lo que la aplicación deberá aceptar ambos formatos de entrada.

En cada paquete que forma el mensaje llega el valor de una serie de campos. Los campos que se requieren para el funcionamiento de la aplicación y que por lo tanto de decodifican son:

- **Categoría:** La categoría de ASTERIX en la que está codificado el paquete.
- **SIC:** Código asociado a cada sistema del aeropuerto. En LEBL el SIC de MLAT es 107 y el SIC de SMR es 007.
- **UTC:** Hora de emisión del mensaje.
- **ICAO Address:** Código único de 24 bits asignado a aeronaves equipadas con transponder modo S.

- **Track Number:** Código que el sistema MLAT asigna a cada aeronave que está detectando. Este número se conserva a no ser que la MLAT no detecte la aeronave en varios segundos seguidos, entonces le asigna un nuevo número de pista ya que lo detecta como una nueva aeronave.
- **Posición:** Coordenadas cartesianas del aeronave referenciadas al centro de coordenadas del sistema MLAT.
- **Velocidad:** Componentes cartesianas de la velocidad del aeronave.

Según el documento ED-117 hace falta del orden de 100.000 paquetes de datos poder obtener datos fiables de los diferentes parámetros a evaluar.

1.3 Proyección de datos WGS-84

En la aplicación se requiere mapear el aeropuerto de Barcelona. Se usa una lista de ficheros que contienen información punto por punto de los diferentes componentes que constituyen LEBL: vías de servicio, pistas de despegue/aterrizaje, terminales… Estos puntos están en formato WGS-84 y surge la necesidad de convertirlo al formato cartesiano para así poderlo implementar junto a los paquetes recibidos del sistema MLAT.

Para pasar del formato WGS84 al formato NED se usa el mismo método usado en **(ver [10])**. Las ecuaciones son las siguientes:

$$x = \frac{a \cdot long}{tmp} \cdot \left[\cos(\phi) - \left(\frac{1-e^2}{tmp^2}\right) \cdot \sin(\phi) \cdot lat\right] \quad \textbf{(1.1)}$$

$$y = \left(\frac{a \cdot (1-e^2)}{tmp^3}\right) \cdot lat + a \cdot \cos(\phi) \cdot \sin(\phi) \cdot \left[\frac{3}{2} \cdot e^2 \cdot lat^2 + \left(\frac{long^2}{2 \cdot tmp}\right)\right] \quad \textbf{(1.2)}$$

$$e^2 = 1 - \left(\frac{b}{a}\right)^2 \quad \textbf{(1.3)}$$

$$tmp = \sqrt{1 - e^2 \cdot \sin^2(\phi)} \quad \textbf{(1.4)}$$

Donde:

$\phi =$ Latitud del origen de coordenadas $\qquad lat =$ Latitud (punto) $- \phi$

$\lambda =$ Longitud del origen de coordenadas $\qquad long =$ Longitud (punto) $- \lambda$

$a = 6378137 \qquad b = 6356752{,}3142$

Siendo el origen de coordenadas de la MLAT: 411749426N 0020442410E

1.4 Recorrido LEBL Mayo 2019

ENAIRE, junto a INDRA, realiza periódicamente recorridos por el aeropuerto de Barcelona con un vehículo terrestre, del cual se monitorea su posición con un GPS diferencial, y además se equipa con un transpondedor modo S para que pueda ser captado por la MLAT. Una vez finalizado el recorrido se comparan los resultados de ambas mediciones y se calculan los diferentes parámetros del documento ED-117.

Para poder comparar los resultados de la aplicación de este trabajo con las ya existentes, se evalúan los mismos ficheros del sistema MLAT. Particularmente se evalúa el fichero más reciente, el del recorrido de Mayo de 2019.

1.4.1 Procedimiento

En este apartado se explica cómo se lleva a cabo la grabación del recorrido del vehículo terrestre.

Primero hay que planear cual será la ruta sobre el mapa, intentando que cubra todas las zonas importantes a evaluar. También se define en qué posición se colocará la estación terrestre que el D-GPS necesita para su correcto funcionamiento. Se ubica en uno de los puntos georreferenciados del aeropuerto.

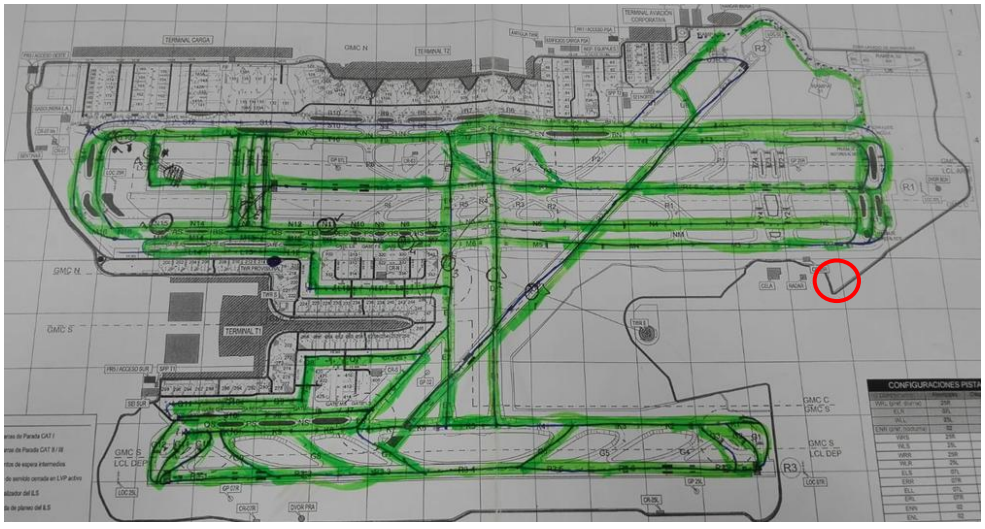

Fig. 1.1 Recorrido planeado en verde; ubicación estación terrestre en rojo

También hay que coordinarse con ATC para acordar a qué hora se realizará el recorrido y que zonas se van a cruzar. También se fija el Call Sign que tendrá asociado el transponder. En nuestro caso es MLAT01. El ICAO Address del transponder que se usa en este caso y que después se usará en la aplicación es 344399.

El siguiente paso requiere que se formen dos equipos: el primero se ocupa de grabar los datos que está generando la MLAT; y el segundo equipo se ocupa de conducir el coche, así como de instalar la estación terrestre en el punto georreferenciado escogido.

Fig. 1.2 Estación terrestre sobre el punto georreferenciado

Por otro lado, en el coche se instala un mástil que hace de antena para el transponder; y una antena con forma de disco para el D-GPS. Se instalan lo más cerca posible en el eje longitudinal para minimizar la diferencia en posición y no añadir error en las medidas.

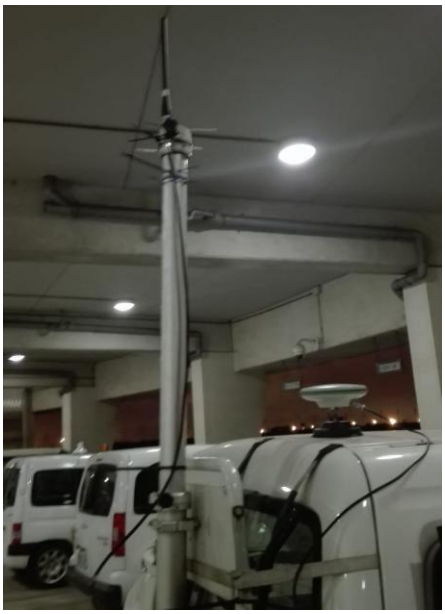

Fig. 1.3 Vehículo equipado antena DGPS y mástil con antena transponder

La tasa nominal del sistema MLAT es de 1s. Para disminuir la distancia en tiempo entre las muestras que se comparan de D-GPS con MLAT, se capturan muestras del D-GPS cada 200ms, asegurando así que siempre se estará comparando la posición de puntos que estén como máximo a ± 200ms.

Dentro del coche se sitúa el transponder de MLAT; el receptor D-GPS y un equipo de radiofrecuencia para comunicarse con ATC.

Fig. 1.4 y Fig. 1.5 Equipo de radiofrecuencia a la izquierda y equipos de MLAT (gris) y D-GPS (blanco) a la derecha

Por otro lado, desde la torre de control se graban los datos que genera la MLAT. Se puede seguir el recorrido que se está realizando a través de la aplicación Visual radar 3000.

Fig. 1.6 En verde, la posición del vehículo que muestra Visual radar 3000

1.4.2 Resultado

Los resultados obtenidos del recorrido, tal y cómo se ven en la interfaz final de la aplicación que se realiza en este trabajo son:

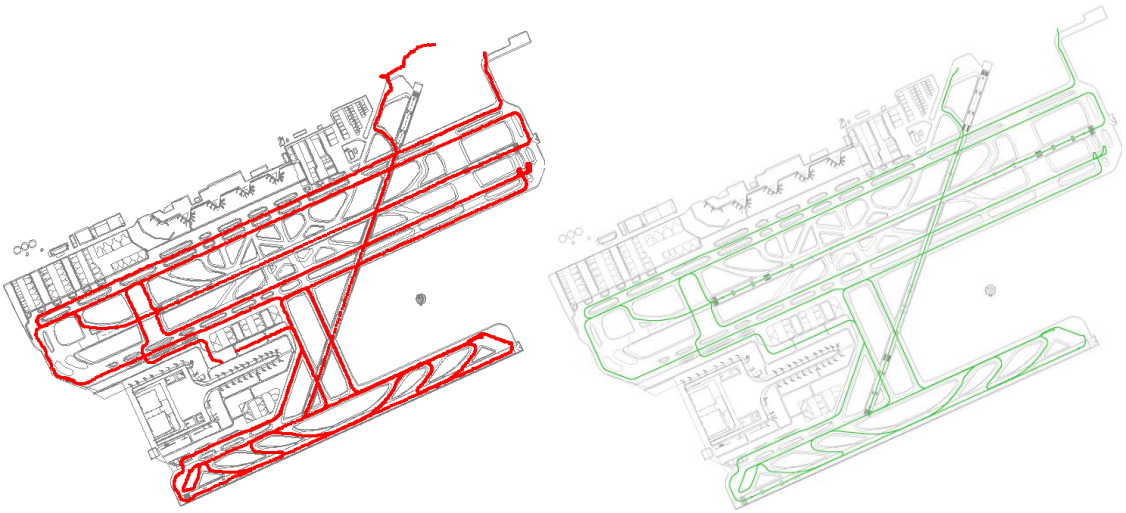

Fig. 1.7 y Fig. 1.8 En rojo el recorrido recogido de la MLAT; en verde del D-GPS

Los resultados de los diferentes parámetros del documento ED-117 respecto a este recorrido se encuentran los capítulos 3 y 4 de este documento.

CAPITULO 2. SEGMENTACIÓN de LEBL

2.1 Zonas requeridas

2.1.1 Especificaciones ED-117

Para evaluar los parámetros del doc. ED-117 es necesario diferenciar entre varias zonas dentro de LEBL. Son las siguientes:

Tabla 2.1. Tipo de áreas definidas en el aeropuerto de Barcelona según clasificación EUROCAE ED-117.

Clasificación	Denominación	Comentarios	LEBL
Área tipo 1	Área de Maniobras	Abarca el área de maniobras del aeropuerto.	Las 3 pistas de aterrizaje/despegue y la zona de taxi.
Área tipo 2	Apron	Abarca las plataformas del aeropuerto.	Diferenciaremos entre Apron de la terminal 1 (T1) y de la terminal 2 (T2).
Área tipo 3	Stands	Abarca las posiciones de stand del aeropuerto.	Diferenciaremos entre Stands de la T1 y la T2.
Área tipo 4	THR-XX (0 – 2,5 NM)	Área de vuelo 1. Área desde el umbral hasta 2,5NM el mismo, y entre +/- 10º del eje de pista, desde el centro de la misma.	Un total de 6, una para cada cabecera del total de 3 pistas.
Área tipo 5	THR-XX (2,5 – 5 NM)	Área de vuelo 2. Área desde 2,5NM hasta 5NM del umbral, y entre +/- 10º del eje de pista, desde el centro de la misma.	Es la continuación de las áreas de tipo 4, por lo que también hay un total de 6.

Dentro de la aplicación también se evaluará el Airborne, el cual será cualquier área que no esté dentro de las zonas definidas en la tabla **(Tabla 2.1)**.

También se filtrará por velocidad para distinguir blancos estáticos o vehículos terrestres de aviones que estén en las áreas de tipo 4, 5 o Airborne.

A continuación se calcula el tamaño de las áreas de tipo 4 y 5, para así poder determinar las coordenadas que las forman en cada cabecera de pista.

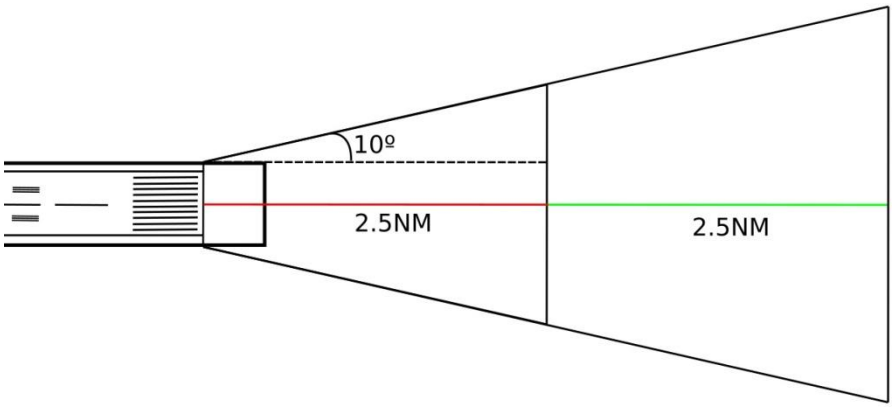

Fig. 2.1 Esquema de la definición de las áreas de tipo 4 y 5

Calculamos el tamaño de sus lados y obtenemos:

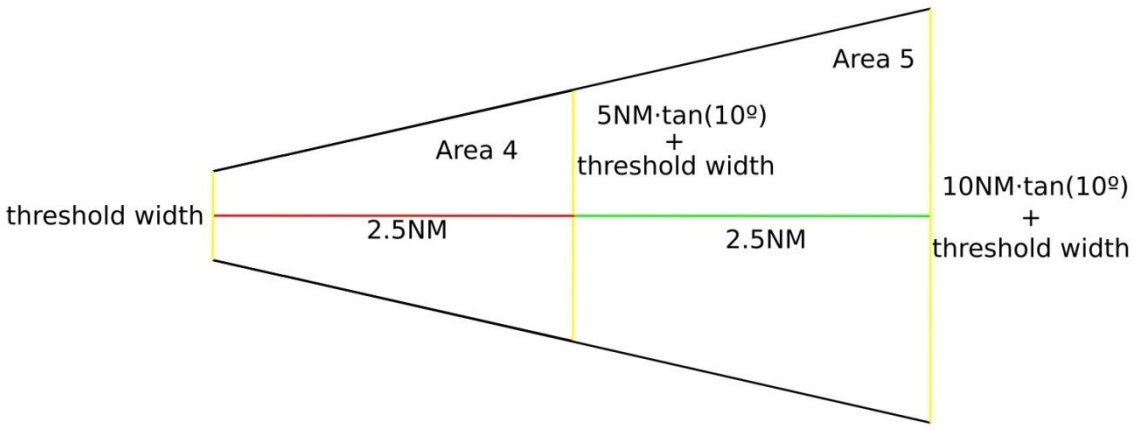

Fig. 2.2 Esquema del tamaño de cada uno de los lados que forma las áreas de tipo 4,5

2.1.2 Segmentación propuesta

En el momento de realizar la segmentación de LEBL se piensa primero en cómo será el algoritmo que clasifique cada punto en su zona correspondiente. Este algoritmo tendrá la información de que área del mapa conforma cada tipo de zona, por lo que comprobará si un punto está dentro de un área siguiendo un orden de prioridad, el cual es el siguiente:

- Runways, en el siguiente orden: pista 02, pista 25L, pista 25R.
- Stands.
- Apron.
- Taxi.

- Área tipo 4.
- Área tipo 5.
- Airborne (el resto).

Esto permite que haya superposición de zonas, reduciendo el número de zonas y su complejidad. Un ejemplo de ello es la zona de taxi (verde en la figura **(Fig. 2.3)**), la cual cubre la mayor parte de las zonas de RWY.

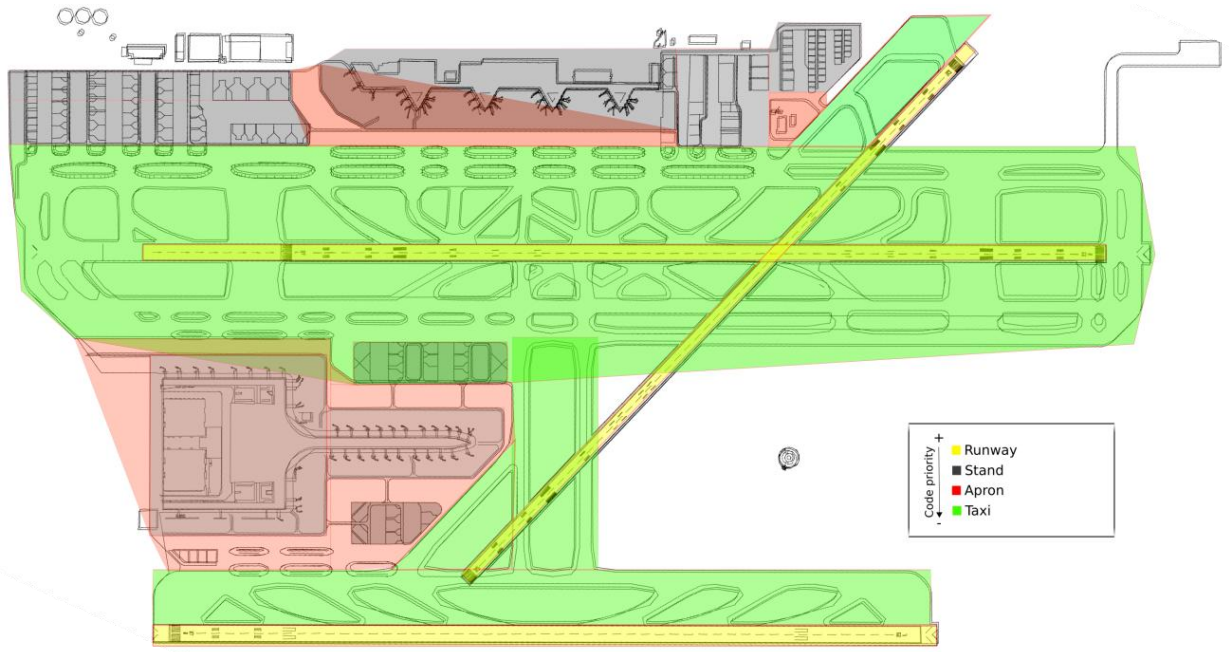

Fig. 2.3 Segmentación propuesta inicialmente para el aeropuerto de Barcelona

La figura **(Fig. 2.3)** aún no incorpora las áreas de tipo 4 y 5.

Para implementar las áreas en el mapa se ha creado un fichero con las coordenadas de cada punto que define los vértices de las diferentes zonas.

2.2 Método matemático

Para poder situar cada paquete en una zona, se aplicada un método de optimización lineal. Con los puntos que definen los vértices de cada zona se crean parejas de puntos que crean inecuaciones.

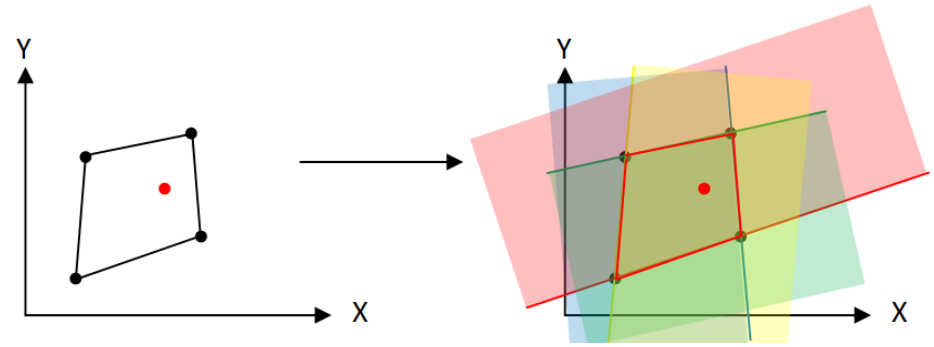

Fig. 2.4 A la izquierda: en rojo el punto cuya zona se quiere determinar, en negro los puntos que forman los vértices de la zona; a la derecha: cada pareja de puntos crea una inecuación

De cada pareja de puntos se obtiene una inecuación, y se comprueba si el punto cumple con todas las inecuaciones de esa sección. La zona debe ser cerrada y convexa.

La inecuación que se quiere obtener es la siguiente:

$$A.y + B.x \geq C \qquad \textbf{(2.1)}$$

Siendo x, y las coordenadas del punto a comprobar. Se parte de:

$$y - Y_1 = m \cdot (x - X_1) \qquad \textbf{(2.2)}$$

Siendo m: $$m = \frac{Y_2 - Y_1}{X_2 - X_1} \qquad \textbf{(2.3)}$$

Siendo P_1, P_2 dos puntos consecutivos de la sección: $P_1(X_1, Y_1)$; $P_2(X_2, Y_2)$

Se obtiene:

$$\underbrace{y}_{A.y} - \underbrace{m.x}_{B.x} = \underbrace{Y_1 - m.X_1}_{C} \qquad \textbf{(2.4)}$$

Falta determinar el signo de la desigualdad. Para ello se tiene en cuenta el ángulo que forma el vector director de la recta respecto la horizontal. Se parte de la premisa de que lo puntos que forman la sección se han introducido en el código de forma anti horaria:

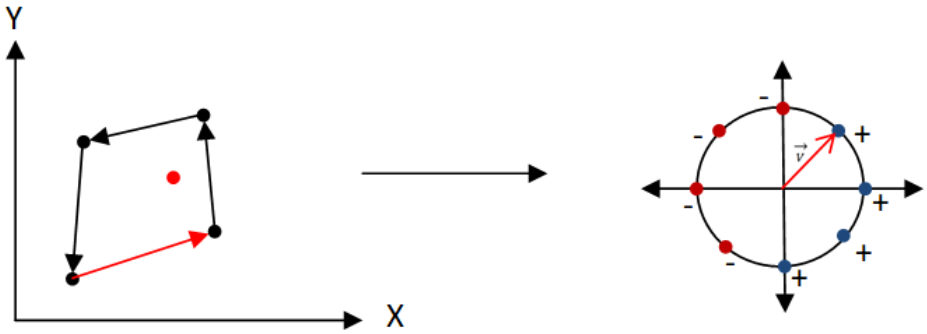

Fig. 2.5 A la izquierda: en rojo la recta que une de forma anti horaria dos puntos consecutivos; a la derecha: el signo de la inecuación en función de la orientación de su vector director (dónde + es ≥ y – es ≤)

2.3 Velocidades límite

Otro punto a tener en cuenta es que aunque un punto caiga en una zona determinada, puede ser que no esté en el aeropuerto sino, por ejemplo, sobrevolándolo. Otro posible caso sería un vehículo terrestre pasando por las áreas de tipo 4 y 5.

Para diferenciar estos puntos, se tiene en cuenta la velocidad del avión. Todos los paquetes que contienen posición, contienen también velocidad.

Se tomará el modelo Airbus A-320 como avión de referencia para poner los límites entre velocidad terrestre y aérea. Es importante tener en cuenta que los aviones que están a punto de despegar y los que acaban de aterrizar tendrán velocidades considerablemente altas.

Velocidades medias A320 (depende de TOW, temperatura…):

- Entre 255 y 290 km/h despegue (~70/81 m/s).
- 885 y 935 km/h crucero (~245/260 m/s).
- 240 a 270 km/h aterrizaje (~67/ 75 m/s).
- 20kt en rodadura. (10,5m/s).

Teniendo en cuenta estos valores y siendo conservadores para no descartar ningún tipo de avión, se han establecido las siguientes velocidades límite para cada zona:

- Para las 3 RWY y áreas de tipo 4 y 5: 120m/s.
- Para el resto del aeropuerto: 35m/s.*
- Airborne: Serán los aviones restantes.

*Inicialmente eran 20m/s per algunos vehículos superaban esta velocidad y eran asignados a Airborne.

2.4 Solución de errores

2.4.1 Cruce de zonas

Surge un problema en el cruce de la pista 25L con la fase final de aproximación de la pista 02, ya que identifica el avión como si estuviera rodando por la 25L.

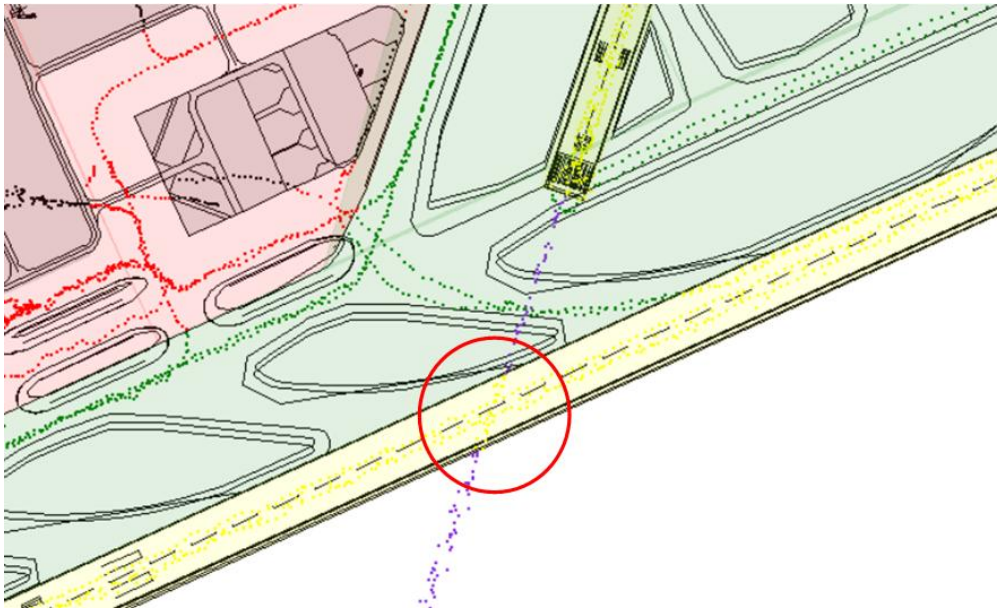

Fig. 2.6 En lila aviones aterrizando en la pista 02, en amarillo aviones rodando por la pista 25L

Lo solucionamos introduciendo una zona dentro de la pista 25L, la cual se analiza en código con prioridad sobre la 25L, la cual asigna los aviones en función de la dirección en la que se mueven. Encontramos la dirección de la pista en el AIP de LEBL:

Tabla 2.2 Características físicas de las pistas 02 y 07R

RWY	Orientación Direction	DIM (m)	THR PSN	THR ELEV TDZ ELEV
02	018.98º GEO 019º MAG	2528 x 45	411715.93N 0020505.41E	THR: 2.0 m / 7ft TFZ: 3.3m / 11ft
07R	065.57º GEO 065º MAG	2660 x 60	411656.32N 0020427.66E	THR: 2.4 m / 8ft TFZ: 3.3m / 11ft

Aproximadamente 19º en la 02 y 65,5º en la 07R (complementaria de la 25L). Pondremos el filtro por dirección en la mitad entre ellas: **42,25º**. Por lo tanto los vuelos aproximándose a la pista 02 serán los que vayan en una dirección geográfica de 19º ± (42.25º - 19º), dando 19º ±23.25º. Para contemplar también posibles despegues de la pista 20, también filtraremos los que vayan en una dirección opuesta: 199º ±23.25º.

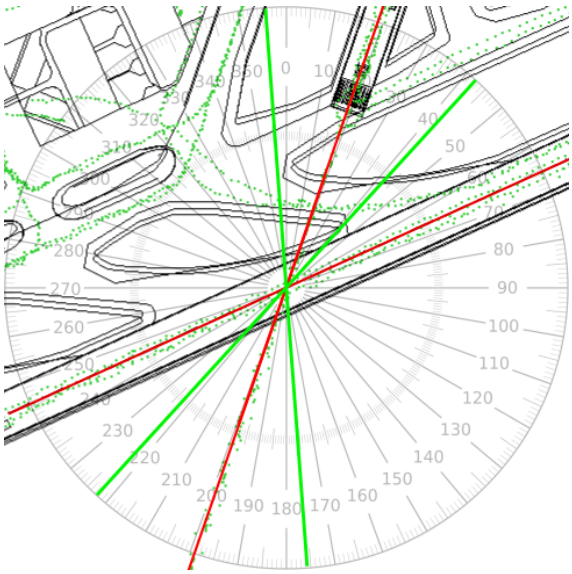

Fig. 2.7 Filtro direccional entre las pistas 02 y 25L

Si filtramos por dirección y no por altura (FL) es porque no todos los paquetes contienen información de altura, y los que la contienen a veces no es fiable o tiene muy poca resolución para determinar si está a unos pocos ft o está rodando en tierra. Nos encontramos con un problema similar en el cruce de la 25R con la 02, pero en este cruce ambos tráficos están rodando por pista:

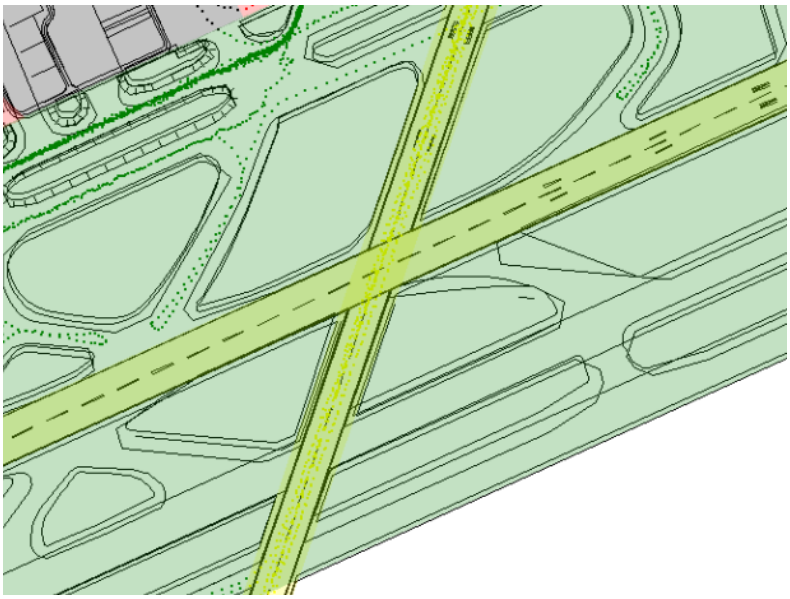

Fig. 2.8 Cruce de las pistas 25R y 02

Por prioridad en el código, todos los aviones que estén en el cruce serán asignados a la pista 02. Solucionamos el problema de la misma manera que en el caso anterior, filtrando por dirección:

Tabla 2.3 Características físicas de las pistas 02 y 07L

RWY	Orientación Direction	DIM (m)	THR PSN	THR ELEV TDZ ELEV
02	018.98º GEO 019º MAG	2528 x 45	411715.93N 0020505.41E	THR: 2.0 m / 7ft TFZ: 3.3m / 11ft
07L	065.57º GEO 065º MAG	3352 x 60	411741.44N 0020419.02E	THR: 2.5 m / 8ft TFZ: 3.5m / 8ft

La orientación de la pista 07L es exactamente la misma que la de la 07R, por lo que los ángulos resultantes también.

Hay algunos puntos en los que no se asigna correctamente la zona debido a que al salir de la pista cambia la dirección. Este error es permisible y pasa en pocos casos.

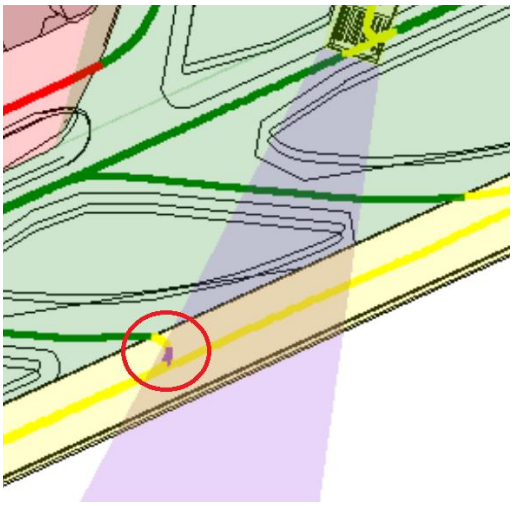

Fig. 2.9 Aeronave sale de la pista 25L y durante un periodo toma la dirección de la fase de aproximación a la pista 02

2.4.2 Tiempo de procesado

Primero se implementó la asignación de una zona a cada paquete de manera individual, por lo que para cada paquete se accedía a los datos de posición de las secciones, se creaban las inecuaciones…Esto aumentó considerablemente el tiempo de procesado en la lectura del fichero (de 5.5 a 25.2 segundos):

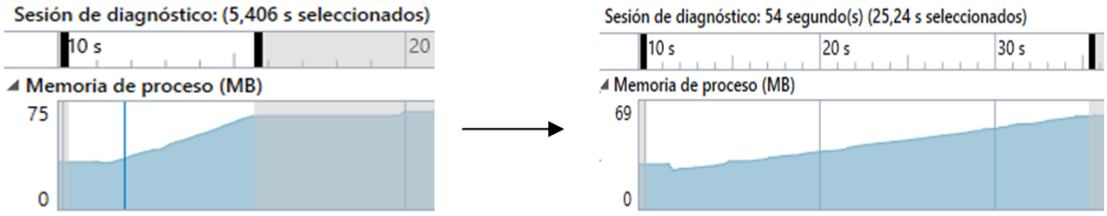

Fig. 2.10 Tiempo de procesado requerido para evaluación del fichero antes y después de implementar la asignación de zonas

Para solucionarlo se clasificó cada paquete en su zona después de haber decodificado el mensaje entero. De esta manera solo se tenía que acceder una vez a los puntos que definen cada zona creando también solo una vez el conjunto de inecuaciones. El tiempo de procesado obtenido al implementar los cambios pasó a ser de 5.6 segundos, solo una decima más que al principio (sin asignación de zonas):

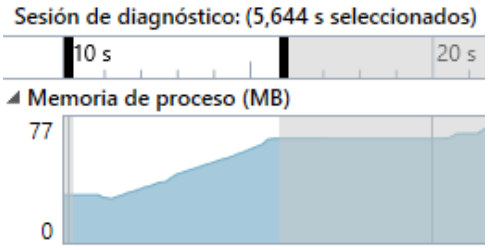

Fig. 2.11 Tiempo de procesado requerido para evaluación del fichero después de reubicar el proceso de asignación de zonas

2.5 Segmentación final

Como el tiempo de procesado no se ve casi afectado por la clasificación de cada paquete, se puede aumentar el número y complejidad de las zonas, respetando siempre el hecho de que tienen que ser convexas y cerradas. En la figura **(Fig. 2.12)** se puede ver la segmentación final de LEBL. Cada número define un área, y cada color define a qué tipo de zona pertenece cada área. En código se comprueba si un punto pertenece a un área determinada de forma ascendente siguiendo la numeración.

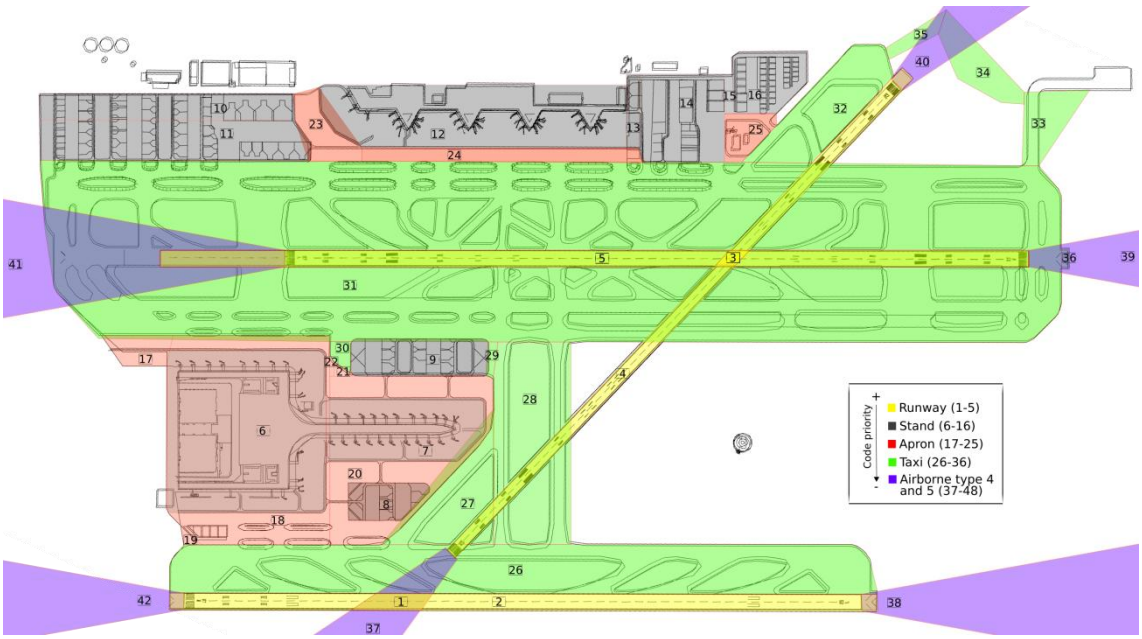

Fig. 2.12 Segmentación final del aeropuerto de Barcelona

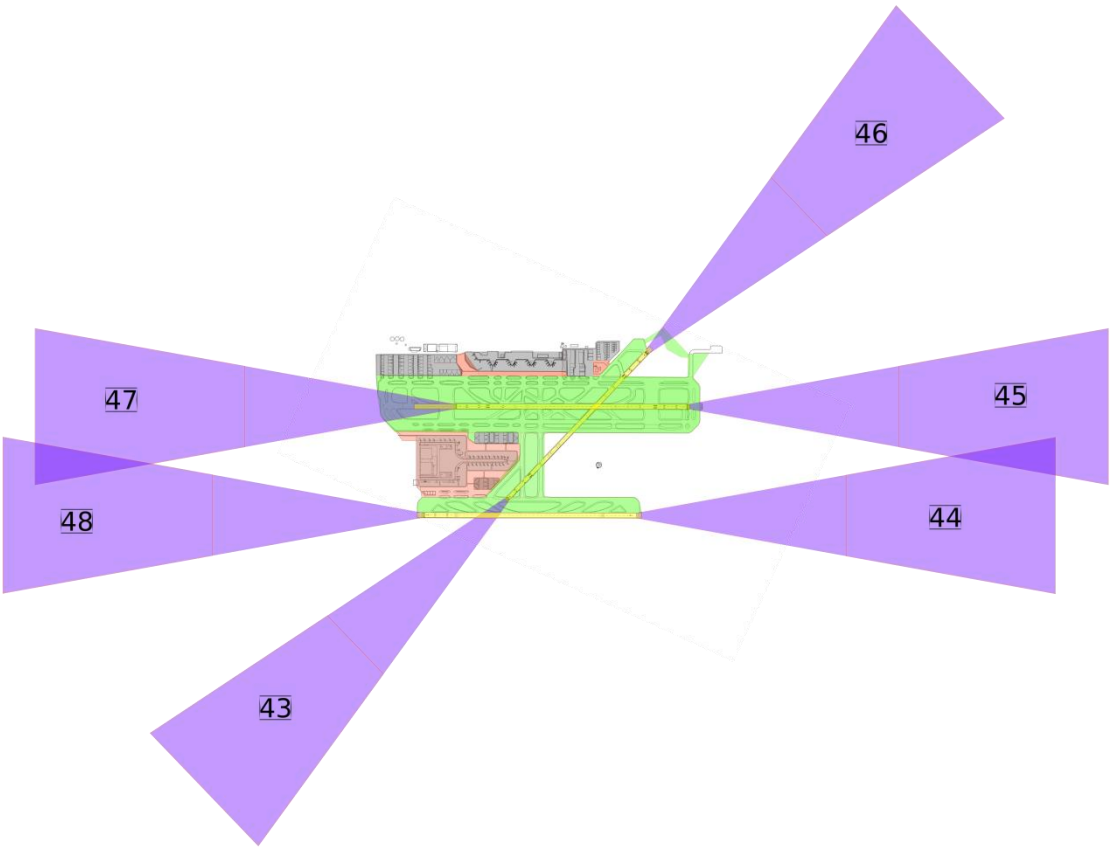

Fig. 2.13 Localización de las áreas de tipo 5 en la segmentación final del aeropuerto de Barcelona

En la figura **(Fig. 2.14)** se puede ver la asignación de zonas que hace la aplicación. El color del punto sigue el mismo código de colores que la figura **(Fig. 2.12)**.

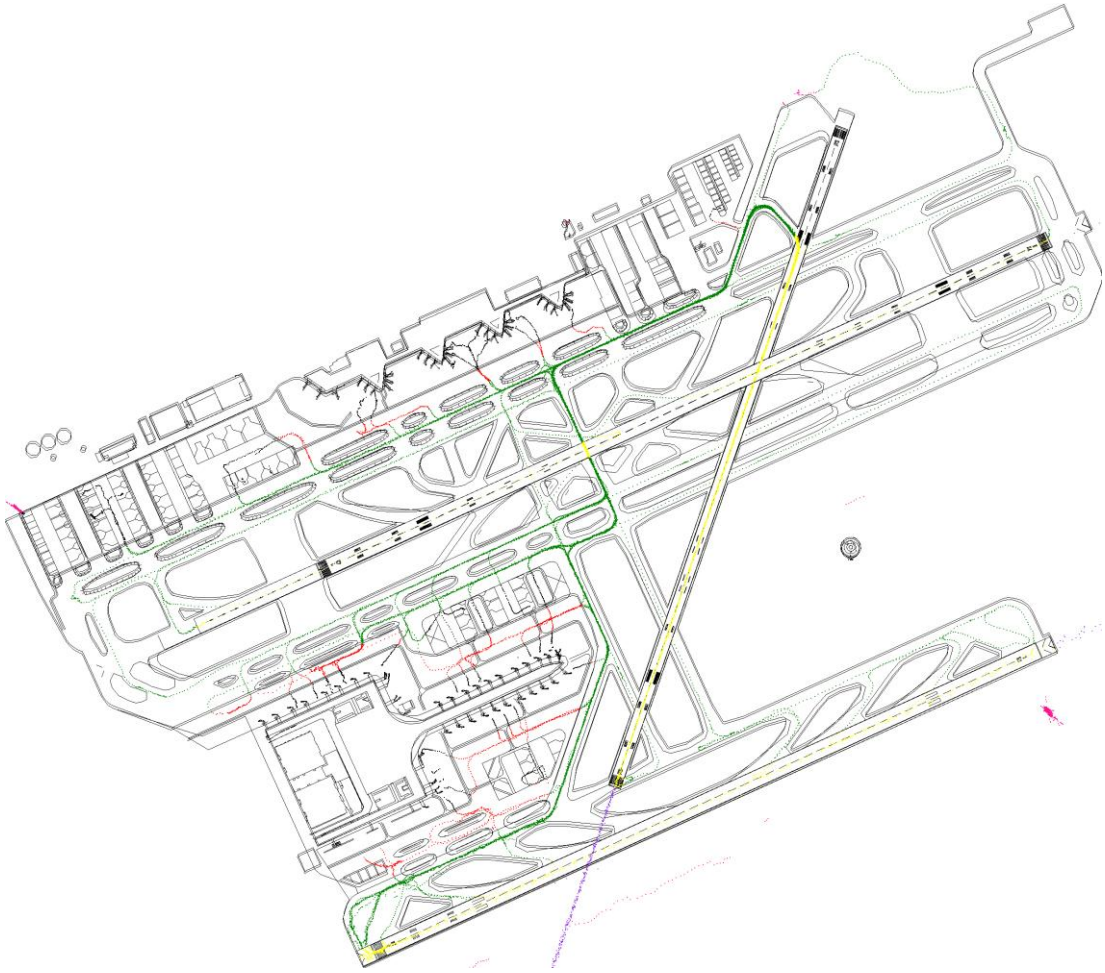

Fig. 2.14 Fichero de ejemplo visualizado desde la aplicación

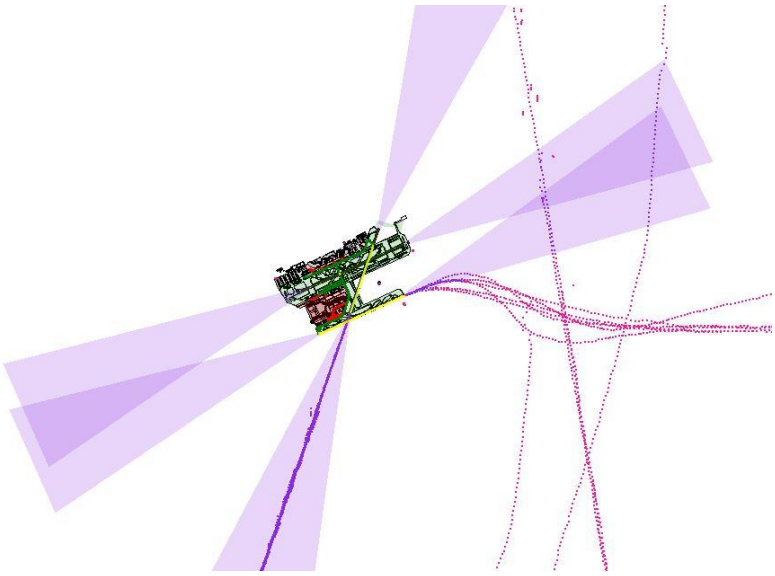

Fig. 2.15 Fichero de ejemplo incluyendo Airborne

CAPITULO 3. Parámetros ED-117

3.1 Introducción

En este capítulo se entrará en detalle en cada uno de los parámetros que será evaluado del sistema MLAT. También se explicará cómo se ha implementado su cálculo en la aplicación y de los problemas que han aparecido durante el proceso, debido mayoritariamente a aspectos desconocidos inicialmente del formato de entrada de los ficheros de la MLAT.

En el doc. ED-117 estos parámetros se definen como las “especificaciones de mínimos de actuación en condiciones estándar”, siendo los siguientes:

- Update Rate (UR).
- Position Accuracy (PA).
- Probability of MLAT Detection.
- Probability of Identification (PID).
- Probability of False Detection (PFD).
- Probability of False Identification (PFI).

3.2 Update Rate

3.2.1 Especificaciones

Este parámetro establece la tasa de refresco mínima que debe tener la información proporcionada por la MLAT para un blanco determinado dentro del área de cobertura. Así mismo establece para cada zona del aeropuerto una probabilidad mínima de que la tasa de refresco supere el límite establecido.

En todo el aeropuerto la tasa de refresco no debe ser superior a 1 segundo en media. La probabilidad asociada a cada zona es la siguiente:

- Apron 70%.
- Stand 50%.
- Área de maniobras 95%.
- Airborne 95%.

3.2.2 Implementación

Para calcular la probabilidad de cada zona, primero se calculan los mensajes que se espera recibir de la MLAT teniendo en cuenta los blancos que se encuentran dentro del área de cobertura y la tasa de refresco media de 1 segundo; y después se recuentan los mensajes que realmente se han recibido por la MLAT.

Para determinar los **refrescos esperados** en cada zona, agrupamos todos los vuelos en función de su ICAO Address y determinamos que zonas cruzan durante su trayecto. Para cada zona cruzada determinamos el punto de entrada (tiempo inicial t_i) y de salida (tiempo final t_f) de esa zona, siendo $(t_f - t_i + 1^*)$ el número de refrescos esperados.

* $(+1)$ para incluir el mensaje inicial en el recuento. Un ejemplo de esto sería:

- $t_i = 2{,}2s;\ t_f = 4{,}2s$
- Refrescos esperados de un blanco determinado en una zona determinada = $4{,}2 - 2{,}2 + 1 = \mathbf{3}$

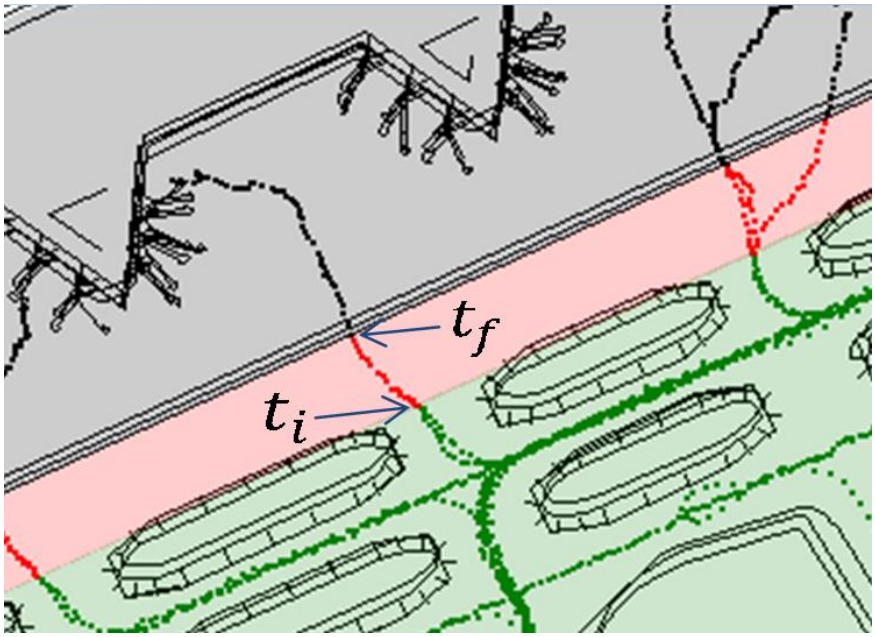

Fig. 3.1 En rojo, aeronave entrando y saliendo de la zona de Apron

Para comprovar la efectividad del metodo comparamos el total de refrescos esperados obtenidos de esta manera con el total de refrescos esperados utilizando el mismo metodo pero sin filtrar por zonas, por lo que solo tenemos en cuenta el tiempo inicial y final de recepción de datos de un avión. Obtenemos 74.103 refrescos esperados y 76.906 respectivamente, lo equivalente a 46min sin mensajes recibidos por dicho blanco. La diferencia entre ambos se debe a un factor que no se ha tenido en cuenta y que se explica en el apartado **(3.2.3 Solución de errores)**. Se hace separando por zonas y no analizando el total del trayecto debido a que cada zona tiene asociada una probabilidad mínima respecto a la tasa de refresco diferente.

Por otro lado, para contar los **refrescos recibidos** usamos el siguiente criterio:

Cada mensaje que llegue 1 segundo después del anterior (para un mismo avion) contará como mensaje recibido. Si en cambio tarda más de 1 segundo no cumplirá y no será contado. A partir de ahí se volverá a contar 1 segundo a partir del mensaje que no ha sido contado.

3.2.3 Solución de errores

La diferencia entre los dos metodos diferentes utilizados en el recuento de refrescos esperados en el apartado **(3.2.2 Implementación)** se debe a lo siguiente:

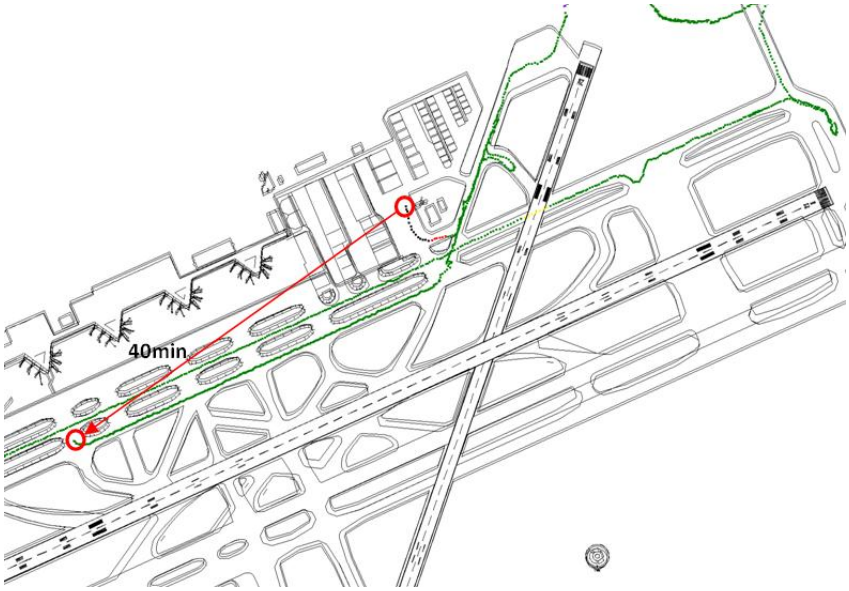

Fig. 3.2 En verde, recorrido entero del vehículo; en rojo un periodo de silencio por parte del vehículo de más de 40min

Esto se debe a que el vehículo terrestre deja de emitir en una zona del aeropuerto y vuelve a emitir 40min más tarde en otra zona distinta. El primer método no tiene en cuenta los mensajes que se deberían haber recibido durante este tiempo, ya que solo considera tiempo inicial y final dentro de una misma zona. En cambio el segundo método si lo tiene en cuenta. En este caso el valor correcto es el que hemos obtenido con el primer método, pero también nos podríamos haber encontrado con que un avión estacione en un Stand, apague todos los sistemas, y en la misma grabación vuelva a encender los sistemas. Un ejemplo de ello se muestra en la figura **(Fig. 3.3)**. Como no ha cambiado de zona el código identificará que todo ese tiempo tendría que haber estado transmitiendo, introduciendo un error entre mensajes esperados y mensajes realmente recibidos.

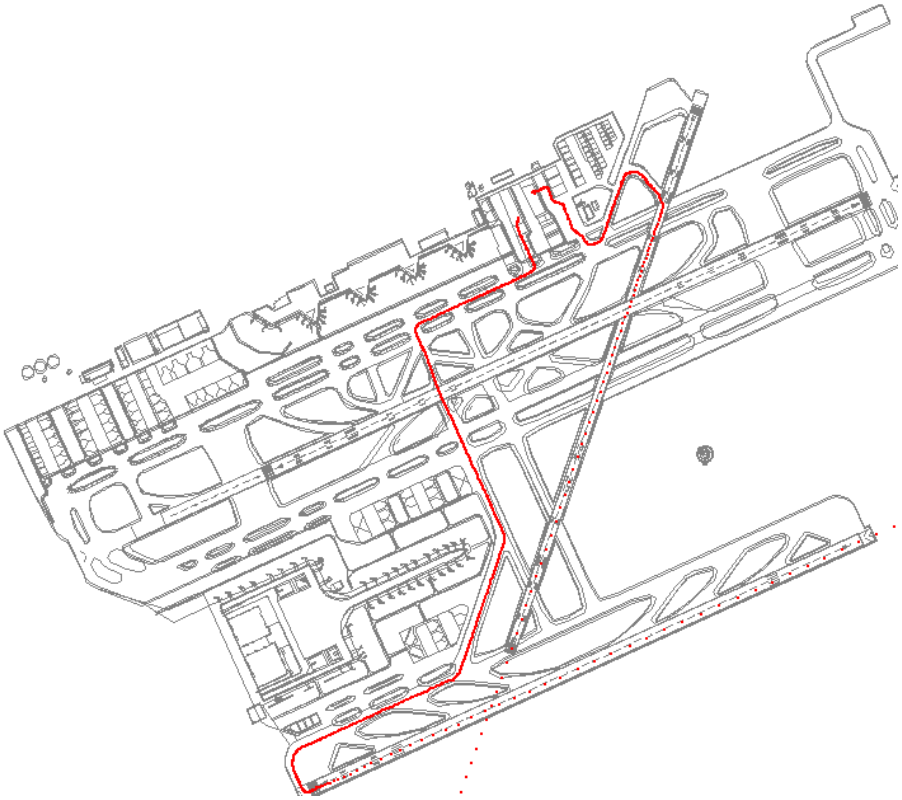

Fig. 3.3 Avión (modelo Gulfstream G650ER) aterriza por la pista 02, estaciona en un Stand de la T2, hace el “turn round” y vuelve a despegar por la pista 07R

Para solucionar el error, consideraremos que si no tenemos datos de un vehículo durante más de **1min**, no es porque el MLAT no lo esté detectando sino porque el vehículo ha dejado de transmitir.

3.2.4 Resultados

Tabla 3.1 Resultados de Update Rate del fichero de verano 2019 con el formato que ofrece la aplicación

	Updates	Expected	UR (%) (updates/s)	Minimum UR (%)
Maneouvering Area	**20625**	**20719**	**99,545**	**95**
RWY 25L	698	719	97,075	
RWY 02	2157	2158	99,953	
RWY 25R	790	791	99,873	
Taxi	16980	17051	99,582	
Apron	**3572**	**3633**	**98,302**	**70**
Apron T1	2916	2970	98,159	
Apron T2	656	663	98,941	
Stands	**8135**	**8166**	**99,609**	**50**
Stands T1	3426	3444	99,451	
Stands T2	4709	4722	99,723	
Airborne	**13486**	**17383**	**77,579**	**95**
Type 4	2457	2478	99,15	
Type 5	2095	2131	98,309	
Rest	8934	12774	69,936	

En la filas se separa en la cuatro areas que requiere el documento ED-117:

- Area de maniobras, que engloba las tres pistas y la zona de taxi.
- Arpon, que se ha desglosado en el apron de cada terminal de LEBL para tener resultados más detallados.
- Stands, también desglosados para cada terminal.
- Airbonre, que incluye las areas de tipo 4 y 5 debido a que en las especificaciones no se hace referencia a ellas.

Aunque el resultado global en Airborne es menor al mínimo, en las zonas de tipo 4, 5 está por encima del mínimo. También es importante remarcar que la MLAT no está diseñada para cubrir la zona de Airborne.

3.3 Position Accuracy

3.3.1 Especificaciones

En este parámetro el documento ED-117 declara un error máximo entre la posición horizontal recibida de un blanco y su posición real, asociado a un porcentaje mínimo de tiempo durante el cual el error debe ser menor al máximo. Una vez más declara valores diferentes para cada zona:

- Área de maniobras y Apron: Error máximo de 7.5 m el 95% del tiempo. Y un error máximo de 12 m el 99% del tiempo.
- Stand: Error máximo de 20 m promediado en periodos de 5 segundos.
- Área de tipo 4: Error máximo de 20 m el 95% del tiempo.
- Área de tipo 5: Error máximo de 40 m el 95% del tiempo.

3.3.2 Implementación

Dado que el objetivo de la aplicación es trabajar con solo tráfico de oportunidad, primero se ha buscado la manera de obtener el valor de este parámetro solo con la información que nos llega de la MLAT *(Capitulo 3.3.2.1)*. Esto dificulta el proceso debido a que no tenemos una fuente de información que nos dé la posición real del blanco (como por ejemplo nos la da el D-GPS), lo cual nos permitiría calcular el error en posición horizontal respecto de la información que nos da la MLAT.

Una vez analizada y posteriormente descartada la opción de calcular este parámetro con trafico de oportunidad, en el capitulo (Capitulo 3.3.2.1) se detalla el proceso y resultados del cálculo con el vehículo equipado con D-GPS.

3.3.2.1 Tráfico de oportunidad

En la categoría 10 de ASTERIX no hay ningún campo que haga referencia al error en posición, por lo que se analizaran archivos de tráfico de oportunidad de las categorías 20 y 21.

En la **categoría 21** se reciben datos del ADS-B. Si se consigue obtener los datos de posición del ADS-B y el error en posición asociado, se podría llegar a determinar el error que se produce en la MLAT. A parte del campo de posición, hay un campo llamado “Figure of Merit”, y dentro de este se encuentra el campo “Position Accuracy”. No es el valor de PA en sí, sino un código descrito en ADS-B MASPS (ED-102A) **(ver [9])**, llamado NUCp (Navigational Uncertainty Categories). En la siguiente versión del documento NUCp pasa a llamarse NIC (Navigational Integrity Categories) y NAC (Navigational Accuracy Categories). La relación entre ambos y el valor equivalente de PA está estipulada en el documento ED-102A.

- NIC proporciona el Rc (containment radius). Da información sobre la integridad.
- NAC proporciona EPU (Estimated Position Uncertainty). Da información de accuracy.

En los ficheros de prueba de categoría 21 de LEBL solo se reciben los códigos NUCp 0 y 8. Para poder obtener valores fiables de PA siguiendo este código, los valores de NUCp deberían cambiar entre sus diferentes posibles valores formando una gaussiana. Al solo obtener dos diferentes valores descartamos este campo como posible fuente de información sobre PA.

El motivo por el cual no se recibe información fiable en este campo se debe a que la MLAT de LEBL está pendiente de ser actualizada. Con la instalación de una tarjeta ADS-B en el radar secundario se podrá obtener información de este campo.

Por otra parte, en la **categoría 20** llega el campo “Position Accuracy”. En este campo llega la dilución de la precisión (DOP) y las desviaciones estándar. El campo que nos interesa es el de desviación estándar.

Se analizan varios ficheros de prueba y se mira en cuántos de ellos llega información en el campo de PA. En alrededor del 95% de mensajes recibidos este campo llega con información.

De esos mensajes con información en el campo de PA se mira el valor que tienen las desviaciones estándar x e y. En todos los mensajes ambas desviaciones estándar **valen 0**. Con esto se concluye la no validez del campo.

Aparte del campo ya existente de PA en la categoría 20, en la misma categoría hay otro campo llamado “Campo reservado”, el cual incluye más información que inicialmente no se previó en el mensaje de ASTERIX.

Dentro de este campo reservado hay un item nuevo para position accuracy, que según las especificaciones es más completo (incluye la desviación estándar en formato WGS-84) y está basado en otro método de cálculo (covarianza en vez de correlación).

Decodificamos el campo reservado con y miramos si nos proporciona más información. **En nuestros ficheros de categoría 20 nunca llega información en el campo reservado.**

Ante la falta de otra fuente de información que nos ayude a calcular la PA, se descarta la opción de evaluar el parámetro con tráfico de oportunidad.

3.3.2.2 Vehículo con D-GPS

Ante la imposibilidad de calcular la PA con tráfico de oportunidad, necesitamos de un vehículo que lleve a bordo un D-GPS y un transponder, para poder captar a la vez datos de MLAT y datos de D-GPS. Los datos de D-GPS después serán post procesados y se usarán como referencia (posición real) para calcular la PA. El problema de usar este método es que por un lado no se pueden obtener datos de Airborne (el vehículo usado es terrestre), y por otro lado que la aplicación no podrá funcionar al 100% con solo trafico de oportunidad. El recorrido que se lleva a cabo con el vehículo es el que se muestra en las figuras (**Fig. 1.7 y Fig. 1.8)**.

Para poder comparar el fichero MLAT con el D-GPS se busca en el fichero MLAT los paquetes que tienen el mismo ICAO Address que el que lleva a bordo el vehículo (344399). Después se mira que franja de tiempo comparten y se fija un tiempo inicial y final de franja compartida. Se pueden distinguir 5 casos diferentes:

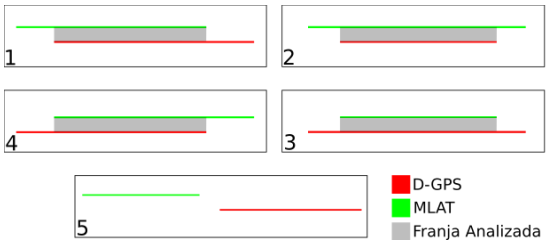

Fig. 3.4 Esquema general de determinación de la franja de tiempo analizable

Finalmente se escoge que puntos de la lista D-GPS están más cerca (en tiempo) de la lista de mensajes MLAT. En principio se tendrá 1 dato/s de MLAT y varios datos/s de D-GPS. En la grabación analizada se reciben 5 datos/s del D-GPS.

Es importante remarcar que el código no diferencia entre posibles ficheros grabados en mismas horas pero días diferentes, debido a la falta de información dentro del mensaje de la MLAT.

Para que no se obtenga mucho error en posición debido a comparar dos puntos grabados en tiempos muy diferentes, se limita la diferencia en tiempo a ±(tiempo de tasa de refresco) del D-GPS. En nuestro caso, si un punto de MLAT no tiene un punto de D-GPS a menos de 0,2s, saltará al siguiente. El código se adapta a la tasa de refresco de cada fichero D-GPS.

3.3.3 Calculo de percentiles

Para calcular el valor máximo de error que se tiene el 95% y 99% del tiempo usaremos los percentiles. Consiste en ordenar una lista de valores de menor o mayor (o viceversa, según sea el objetivo) y ver qué valor tiene el componente que equivale al X% de la lista, siendo ese el valor máximo que tiene la lista el X% de valores de su total.

Definimos percentil (P_i) cómo:

$$P_i = x_{\left(\frac{n\cdot i}{100}\right)+1} \qquad \text{si se cumple que: } \frac{n\cdot i}{100} \notin \mathbb{N} \qquad \textbf{(3.1)}$$

$$\text{Y si en cambio: } \frac{n\cdot i}{100} \in \mathbb{N} \text{ , entonces: } P_i = \frac{x_{\left(\frac{n\cdot i}{100}\right)} + x_{\left(\frac{n\cdot i}{100}\right)+1}}{2} \qquad \textbf{(3.2)}$$

Siendo $\{x_1, x_2, \dots, x_n\}$ el conjunto de datos de error en posición ordenados de menor a mayor e $\{n, i\}$ el numero de datos y el percentil (%) respectivamente.

Para ordenar de menor a mayor el conjunto de datos utilizamos el método “Bubble sort”. Es un algoritmo que funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. En la figura **(Fig. 3.5)** se muestra su funcionamiento en el formato de código de C# (lenguaje que se ha usado para desarrollar la aplicación).

```
int temporal;
for (int i=1; i<=vector.Length; i++)
{
        for (int j=0; j<=vector.Length - 1; j++)
        {
                if (vector[ j ] > vector[ j+1])
                {
                        temporal = vector[ j ];
                        vector[ j ] = vector[ j + 1];
                        vector[ j + 1] = temporal;
```

Fig. 3.5 Método “Bubble Sort” basado en C#

3.3.4 Resultados

Tabla 3.2 Resultados de Position Accuracy del fichero de verano 2019 con el formato que ofrece la aplicación

	P95	P99	Max Detected	max P95	max P99	Mean	STD
Maneouvering Area	7,48	11,82	13	7,5	12	3,61	5,14
RWY 25L	9,9	19,06	11			4,62	6,42
RWY 02	6,83	13,71	12			2,51	4,18
RWY 25R	5,9	6,43	13			3,28	4,62
Taxi	7,73	11,92	4			3,62	5,14
Apron	8,26	10,26	31	7,5	12	4,97	6,57
Apron T1	8,26	10,26	31			4,97	6,57
Apron T2							
Stands							
Stands T1							
Stands T2							
Airborne (No Data)							

No hay datos de Stands ni Apron en T2 porque el vehículo no pasó por esa zona. Aparte, el único percentil que es mayor al máximo es el P95 en la zona de Apron, dónde vale 8.26 m siendo el máximo 7.5. También se calculan para dar más información la media y la desviación estándar.

En la figura **(Fig. 3.6)** se muestran superpuestos las trayectorias grabadas por la MLAT y por el D-GPS. La línea que une cada pareja de puntos de ambos sistemas tiene un color diferente en función de la distancia que los separa. Dentro de la aplicación está la opción de obtener este mapa.

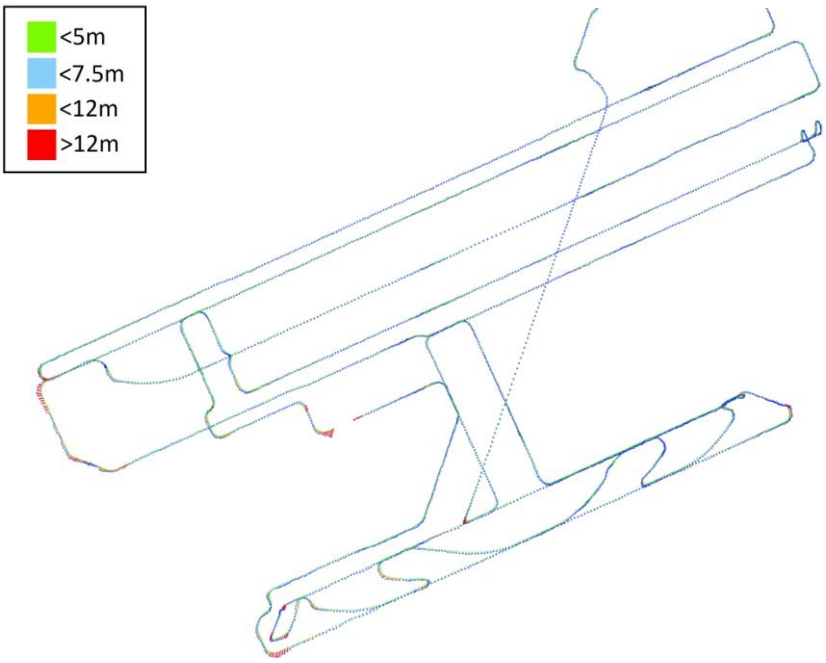

Fig. 3.6 Mapa de la superposición de trayectorias MLAT y D-DGPS

La aplicación también permite exportar los datos de error en x,y a Excel, y con ello se puede obtener el gráfico Scatter de la figura **(Fig. 3.7)**.

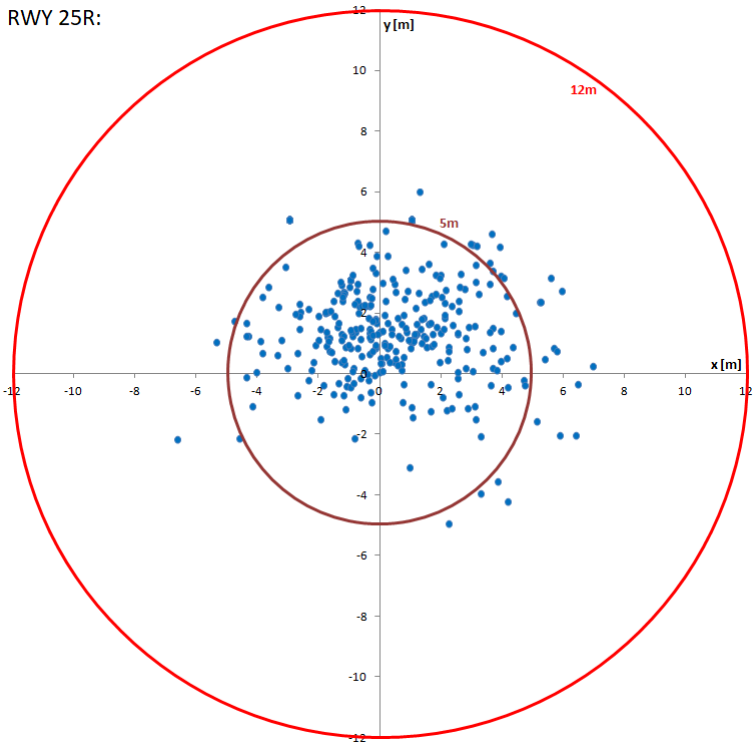

Fig. 3.7 Scatter de los errores en posición en los ejes x,y de la RWY 25R.

3.4 Probability of MLAT Detection

3.4.1 Especificaciones

Un blanco que disponga de transponder modo S y que se encuentre dentro de la zona de cobertura de la MLAT debe reportar su posición, con una probabilidad que depende de la zona aeroportuaria en la que se encuentre:

- Area de maniobras y Apron: Probabilidad de detección superior al 99.9% en cualquier periodo de 2 segundos.
- Stands: Probabilidad de detección superior al 99.9% en cualquier periodo de 2 segundos.
- Airbone: Sin especificar. No requerido.

3.4.2 Implementación

En el código de la aplicación se aísla por ICAO Address y se secciona por zonas, se crean ventanas de 2 segundos (o 5 en función de la zona) y se mira si se ha recibido una señal en esa ventana de tiempo (todas las señales con ICAO Address contienen posición en los mensajes de la MLAT).

Consideramos que un blanco debe ser detectado en la franja de tiempo entre su primera y última detección en una zona determinada, teniendo en cuenta que un vehículo puede apagar el transponder y volverlo a encender al cabo de un rato en la misma zona, usamos el mismo criterio de 1min que en el parámetro de Update Rate del apartado **3.2.**

El total de ventanas son las detecciones esperadas, y las ventanas que cumplen son los casos positivos, siendo la probabilidad de detección los casos positivos entre los casos esperados.

3.4.3 Resultados

Tabla 3.3 Resultados de Probability of MLAT Detection del fichero de verano 2019 con el formato que ofrece la aplicación

	Detected	Expected	PD (%) (Prob. of Detection)	Minimum PD (%)
Maneouvering Area	**23909**	**23931**	**99,908**	**99**
RWY 25L	699	699	100	
RWY 02	2067	2067	100	
RWY 25R	735	735	100	
Taxi	16875	16875	100	
Apron T1	2896	2918	99,246	
Apron T2	637	637	100	
Stands	**7919**	**7926**	**99,912**	**99**
Stands T1	3317	3324	99,789	
Stands T2	4602	4602	100	
Airborne (Not Required)				

Las conclusiones en detalle de los resultados de cada parámetro se harán en el capítulo 5.

3.5 Probability of Identification

3.5.1 Especificaciones

El sistema MLAT debe proporcionar correctamente la identidad de un blanco con una probabilidad superior al 99,9%, esto significa que tiene que detectar correctamente su ICAO Address siempre y cuando el blanco lo esté enviando correctamente. En este parámetro no se diferencia por zonas aeroportuarias.

3.5.2 Implementación

Para identificar un mensaje con un ICAO Address incorrecto utilizaremos el campo “Track Number”. A cada blanco se le asigna un numero de pista (Track number) independientemente de su ICAO Address y lo conserva durante todo el recorrido dentro del rango de la antena. Este número de pista no se asigna según el identificador del vehículo, sino por su posición relativa a la anterior posición. Este número nos permite compara todos los ICAO Address de un mismo número de pista y ver si para un mismo número de pista hay algún ICAO Address diferente al resto (erróneo).
Eurocontrol define en las especificaciones ASTERIX el item Track Number como: “Valor entero que representa una referencia única a una trayectoria dentro de un mismo fichero. “

Nos basamos en eso para suponer que no hay dos aviones a lo largo de la grabación con un Track Number igual, lo cual daría como resultado un gran error en los cálculos del código. Aún así esto no es del todo cierto, ya que la asignación de Track Number depende del fabricante del MLAT. Podría asignarse un número nuevo a cada objetivo hasta llegar al valor máximo y volver a empezar en 0, o podrían reasignarse los números de pista que vayan quedando libres. En LEBL, para un muestreo de unos 100.000 paquetes este hecho no supone un problema.

EUROCAE no especifica en que zonas se requiere este mínimo, por lo que se aplica a todo el tráfico y se desglosa de igual manera que en Update Rate.

Los casos totales son el número de paquetes que contienen Track Number, y los casos correctos son aquellos en que ICAO Address es el mismo que la mayoría de ICAO Address que se encuentran junto a ese Track Number.

3.5.3 Solución de errores

El único problema aparece cuando no hay señal de un blanco en varios refrescos, donde la antena asigna un número de pista nuevo al blanco, ya que no tiene posiciones anteriores (cercanas en tiempo) del blanco. Aún así esto no supone un problema, ya que el código solo compara números de pista iguales con sus correspondientes ICAO Address del paquete, independientemente de si ese número de pista es el mismo que el primer número de pista asignado al vehículo. De hecho esto elimina el problema en que un vehículo apaga el transponder y lo vuelve a encender al cabo de un rato en la misma zona, ya que en ese caso el Track Number se pierde.

Otro problema que no se puede solucionar es el referente al tiempo de procesado. Para cada ICAO Address hay varios Track Number, y tener que agrupar y comparar los Track Number añade tiempo al tiempo de procesado (alrededor de 15 segundos, en función del tamaño del fichero).

3.5.4 Resultados

Tabla 3.4 Resultados de Probability of Identification del fichero de verano 2019 con el formato que ofrece la aplicación

	Correct	Inorrect	PID (%) (correct/total)	Minimum PID (%)
Maneouvering Area	**20718**	**0**	**100**	
RWY 25L	719	0	100	
RWY 02	2158	0	100	
RWY 25R	791	0	100	
Taxi	17050	0	100	
Apron	**3604**	**0**	**100**	
Apron T1	2943	0	100	
Apron T2	661	0	100	
Stands	**8153**	**0**	**100**	
Stands T1	3433	0	100	
Stands T2	4720	0	100	
Airborne	**13812**	**0**	**100**	
Type 4	2477	0	100	
Type 5	2114	0	100	
Rest	9221	0	100	
Total	**46287**	**0**	100	**99**

3.6 Probability of False Detection

3.6.1 Especificaciones

El sistema MLAT solo puede reportar blancos falsos con una probabilidad inferior al 0.01%. El documento ED-117 no hace distinciones entre zonas para este parámetro, pero si excluye la zona de Airborne.

Se define blanco falso como aquel reporte de blanco modo S con error horizontal asociado mayor a 50m.

Para tener una idea de la PFD en áreas de vuelo, se hace también un estudio de este parámetro para las áreas de tipo 4 y 5 de cada cabecera.

3.6.2 Implementación

Como no disponemos de un error horizontal asociado, en el código miraremos si un punto está a más de 50m del punto anterior, siempre y cuando el punto anterior se haya reportado en el segundo de tiempo anterior.

Si un punto está más de 50m del anterior, se considerará falsa detección. Probablemente este punto también esté a más de 50m del siguiente, pero este segundo caso no se contará como falsa detección debido a que esto haría que la mayoría de las falsas detecciones fueran detectadas dos veces. El código no contempla, en consecuencia, dos falsas detecciones seguidas.

Si en la evaluación del fichero se adjunta una ruta con D-GPS, los cálculos se harán comparando parejas de puntos en el mismo instante de tiempo, ya que es un método más fiable, aunque no proporciona datos de las zonas de tipo 4 y 5 de Airborne.

Para las áreas de tipo 4 y 5 se hace un paralelismo con la zona aeroportuaria, donde la distancia máxima para considerar que un punto es falso es de 50 m. Se sigue el método utilizado en el documento de la división NYVI **(ver [8])** sobre la MLAT de Barcelona, en el cual se considera como falsa detección aquella con error posicional asociado mayor que aproximadamente 4 veces el máximo de los errores instantáneos admitido (12m) (~50m). Para las áreas de vuelo se ha seguido el mismo criterio, siendo la distancia umbral para considerar un punto como falso, 4 veces el error posicional instantáneo permitido en cada área. Así, para el área de tipo 4 será de 80m; y para el área de tipo 5 será de 160m.

3.6.3 Solución de errores

La manera correcta de aplicar el criterio de distancia máxima es comparar el punto recibido con el punto real, obtenido por ejemplo con un D-GPS. En nuestro caso no disponemos de los valores reales de posición, por lo que se compara el punto actual con el anterior, por lo que a parte de los 50m de diferencia, está la distancia que recorre el vehículo en esa unidad de tiempo. Los valores umbral que se han estipulado anteriormente no contemplan esto, por lo que no son suficientemente grandes y por lo tanto asigna una señal como falsa en muchos casos. La tabla **(Tabla 3.5)** muestra los resultados sin tener en cuenta este problema. La figura **(Fig. 3.8)** muestra un esquema del método utilizado.

Tabla 3.5 Resultados de probability of false detection de un fichero de prueba con el método inicial

	Reports	False Reports	PFD (%) (False/Reports)	Minimum PFD (%)
Maneouvering Area	**7458**	**231**	**3,097**	
RWY 25L	523	89	17,017	
RWY 02	983	142	14,446	
RWY 25R	61	0	0	
Taxi	5891	0	0	
Apron	**1592**	**0**	**0**	
Apron T1	1401	0	0	
Apron T2	191	0	0	
Stands	**13921**	**0**	**0**	
Stands T1	11478	0	0	
Stands T2	2443	0	0	
Airborne	**2111**	**157**	**7,437**	
Type 4	1164	150	12,887	
Type 5	947	7	0,739	
Total	**25082**	**388**	**1,547**	**0,01**

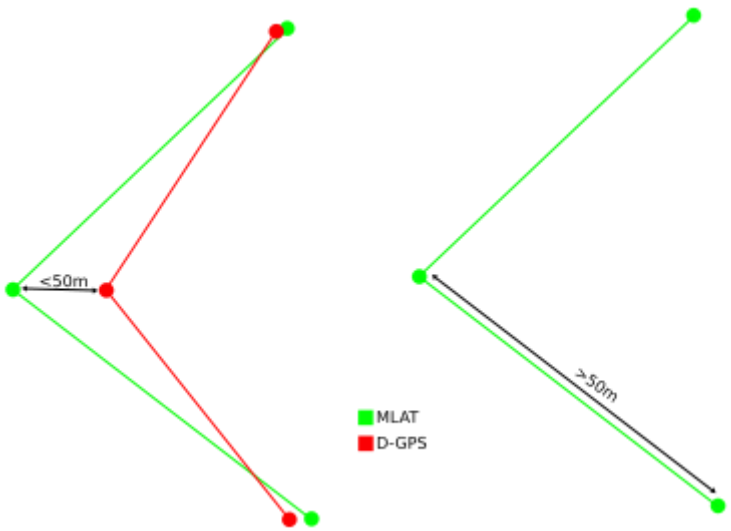

Fig. 3.8 A la izquierda, el método utilizado en caso de disponer de un fichero con datos D-GPS; a la derecha el método utilizado inicialmente para calcular la probabilidad de fala detección con tráfico de oportunidad

Para adaptar el umbral y solo detectar casos reales de falsa detección, hay que incluir en las distancias máximas el desplazamiento del vehículo en 1s de tiempo. Por ello, para cada posición, se comparará el punto actual con un punto ficticio. Este punto ficticio es el resultado de aplicar el vector velocidad sobre el punto anterior durante el periodo de 1s. El umbral máximo estipulado por el doc. ED-117 se aplicará a la distancia entre este punto ficticio y el punto recibido de la MLAT. La aceleración, la cual también se recibe en el mensaje de MLAT, no se tendrá en cuenta para calcular la posición del punto ficticio. La figura **(Fig. 3.9)** muestra un esquema del método utilizado finalmente con tráfico de oportunidad.

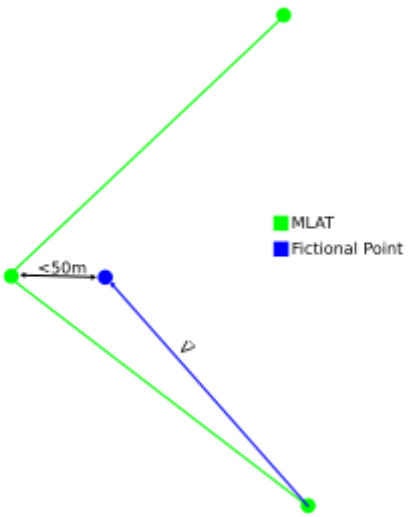

Fig. 3.9 Método utilizado finalmente. El punto azul representa el punto ficticio, que se obtiene de aplicar el vector velocidad al punto anterior de la MLAT

3.6.4 Resultados

Una vez solucionado el error se obtienen los resultados de la tabla **(Tabla 3.6).**

Tabla 3.6 Resultados de probability of false detection de un fichero de prueba con el método final

	Reports	False Reports	PFD (%) (False/Reports)	Minimum PFD (%)
Maneouvering Area	**7458**	**0**	**0**	
RWY 25L	523	0	0	
RWY 02	983	0	0	
RWY 25R	61	0	0	
Taxi	5891	0	0	
Apron	**1592**	**0**	**0**	
Apron T1	1401	0	0	
Apron T2	191	0	0	
Stands	**13921**	**0**	**0**	
Stands T1	11478	0	0	
Stands T2	2443	0	0	
Airborne	**2111**	**10**	**0,474**	
Type 4	1164	9	0,773	
Type 5	947	1	0,106	
Total	**25082**	**10**	**0,04**	**0,01**

Obtenemos 0 detecciones falsas en toda la zona aeroportuaria. En cambio hay 10 detecciones falsas en las zonas 4 y 5 de Airborne.

Las tablas mostradas en el resto de parámetros (las del fichero de verano de 2019) no se corresponden con el fichero que se ha analizado en este caso. La tabla **(Tabla 3.7)** muestra los resultados correspondientes al fichero de verano de 2019.

Tabla 3.7 Resultados de probability of false detection del fichero de verano de 2019

	Reports	False Reports	PFD (%) (False/Reports)	Minimum PFD (%)
Maneouvering Area	**20718**	**0**	**0**	
RWY 25L	719	0	0	
RWY 02	2158	0	0	
RWY 25R	791	0	0	
Taxi	17050	0	0	
Apron	**3604**	**0**	**0**	
Apron T1	2943	0	0	
Apron T2	661	0	0	
Stands	**8153**	**0**	**0**	
Stands T1	3433	0	0	
Stands T2	4720	0	0	
Airborne	**4591**	**4**	**0,087**	
Type 4	2477	3	0,121	
Type 5	2114	1	0,047	
Total	**37066**	**4**	**0,011**	**0,01**

Por otro lado la tabla **(Tabla 3.8)** muestra los resultados correspondientes al fichero de verano de 2019 usando el fichero de D-GPS.

Tabla 3.8 Resultados de probability of false detection del fichero de verano de 2019, pero solo del vehículo de pruebas

	Reports	False Reports	PFD (%) (False/Reports)	Minimum PFD (%)
Maneouvering Area	**6031**	**0**	**0**	
RWY 25L	307	0	0	
RWY 02	174	0	0	
RWY 25R	573	0	0	
Taxi	4977	0	0	
Apron	**1243**	**0**	**0**	
Apron T1	1243	0	0	
Apron T2	0	0		
Stands	**0**	**0**		
Stands T1	0	0		
Stands T2	0	0		
Airborne	**3**	**0**	**0**	
Type 4	3	0	0	
Type 5	0	0		
Total	**7277**	**0**	**0**	**0,01**

3.7 Probability of False Identification

3.7.1 Especificaciones

La probabilidad de que la MLAT detecte erróneamente un blanco debe ser inferior al 0.0001% sobre un promedio de 5 segundos. En este parámetro no se diferencia por zonas aeroportuarias.

Se considera falsa identificación cuando el sistema MLAT identifica el ICAO Address de un blanco erróneamente cuando este la está transmitiendo correctamente.

3.7.2 Implementación

En el código de la aplicación se aísla por vuelo y se secciona por zonas, se crean ventanas de 5 segundos y se mira si se ha recibido un ICAO Address erróneo en esa ventana de tiempo. Para encontrar los casos erróneos se aplica el mismo método que con el parámetro de probabilidad de identificación en el apartado **3.5**, solo que promediando con ventanas de 5 segundos.

El total de ventanas son los periodos totales, y las ventanas que no cumplen son los casos negativos, siendo la probabilidad de falsa identificación los casos negativos entre los casos esperados.

3.7.3 Resultados

Los valores totales deberían ser parecidos a los valores totales del parámetro MLAT Detection, concretamente en el área de stands donde la ventana de

tiempo es la misma. Como se puede ver en la tabla **(Tabla 3.9)** difiere un poco debido a que el recuento se hace a partir del Track Number, y no del ICAO Address.

Tabla 3.9 Resultados de probability of false identification del fichero de verano de 2019 con el formato que ofrece la aplicación

	Total	False	Prob. False ID (%) (false/total)	Minimum Prob. False ID (%)
Maneouvering Area	**18390**	**0**	**0**	
RWY 25L	1773	0	0	
RWY 02	5937	0	0	
RWY 25R	5030	0	0	
Taxi	5650	0	0	
Apron	**4475**	**0**	**0**	
Apron T1	2534	0	0	
Apron T2	1941	0	0	
Stands	**10616**	**0**	**0**	
Stands T1	4148	0	0	
Stands T2	6468	0	0	
Airborne	**10204**	**0**	**0**	
Type 4	2311	0	0	
Type 5	4010	0	0	
Rest	3883	0	0	
Total	**43685**	**0**	**0**	**0,0001**

CAPITULO 4. Vehículos descartados

4.1 Vehículos Squitter

Inicialmente, cuando se prueba la aplicación con algunos ficheros de prueba da unos valores muy bajos en el parámetro de Update Rate. Buscando cual puede ser el error en el código se comprueba que no es un error, sino que hay una serie de vehículos que realmente transmiten en unas tasas muy reducidas, distantes de ser las nominales para MLAT.

Un ejemplo de ello es el vehículo de la figura **(Fig. 4.1).** Su ICAO Address es 345415 y por la ruta que sigue se podría tratar de un vehículo de mantenimiento o vigilancia. En el fichero evaluado envía un total de 5301 mensajes, de los cuales 4248 con separación de un segundo (la nominal); 244 con separación de dos segundos; 638 con separación de tres segundos; 57 con separación de cuatro segundos y 114 con separación de entre 10 y 60 segundos.

En el mismo fichero hay 6 vehículos parecidos a este, y empeoran considerablemente los resultados de la evaluación.

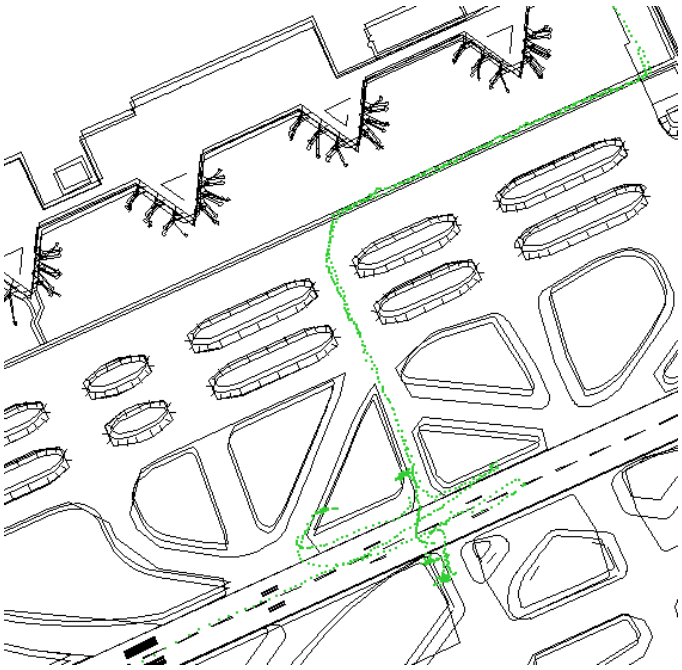

Fig. 4.1 Recorrido del vehículo 345415 en un fichero del verano de 2017

Tabla 4.1 Resultados de Update Rate con el fichero de prueba

	Updates	Expected	UR (updates/s)	Minimum UR
Maneouvering Area	**51574**	**75797**	**0,68**	**0,95**
RWY 25L	1603	1640	0,977	
RWY 02	2712	2730	0,993	
RWY 25R	11402	22104	0,516	
Taxi	35857	49322	0,727	
Apron	**5765**	**5957**	**0,968**	**0,7**
Apron T1	3338	3422	0,975	
Apron T2	2427	2535	0,957	
Stands	**34755**	**35230**	**0,987**	**0,5**
Stands T1	16560	16712	0,991	
Stands T2	18195	18518	0,983	
Airborne	**62955**	**68958**	**0,913**	**0,95**

Este vehículo es uno de los vehículos de Aena que usa squitter en vez de transponder. Estos squitters se comportan distintos de los transponders. Los transponders solo responden si son interrogados por alguna de las estaciones interrogadoras de la MLAT y deben hacerlo cada segundo. En cambio los squitters emiten Modo S de manera autónoma, sin necesidad de ser interrogados y lo hacen con una tasa distinta según se estén moviendo o según estén parados y con tasas distintas según envíen posición o identificación.

Para el cálculo de todos los parámetros ED-117 estos vehículos se eliminan de la grabación, ya que solo se evaluaran los transponders.

4.2 Vehículos Aena

Estos son los vehículos squitters que se usan en LEBL, en fondo blanco de Aena y fondo azul ENAIRE:

Tabla 4.2 ICAO Address de los vehículos que usan Squitter.

341041	34104F	341097	341103	34434B	344357	344389	3443C6
341042	341050	3410C9	341105	34434C	344358	34438A	3443C3
341043	341051	3410CA	341107	34434D	344359	34438B	3443C4
341044	341052	3410CE	341108	34434E	34435A	34438C	3443C5
341045	341053	3410D0	341109	34434F	344381	34438D	3443C2
341046	341055	3410D2	34110A	344350	344382	34438E	3443C6
341047	341081	3410D3	34110B	344351	344383	34438F	3443C3
341049	341082	3410D4	34110F	344352	344384	344390	3443C4
34104B	341090	3410D5	344347	344353	344385	3443C3	3443C5
34104C	341092	3410D6	344348	344354	344386	3443C4	3443C2
34104D	341094	3410D7	344349	344355	344387	3443C5	3443C6
34104E	341095	3410D8	34434A	344356	344388	3443C2	

Los resultados de Update Rate después de descartar estos vehículos son:

Tabla 4.3 Resultados de Update Rate con el fichero de prueba después de eliminar los vehículos que usan Squitter

	Updates	Expected	UR (updates/s)	Minimum UR
Maneouvering Area	**21292**	**21433**	**0,993**	**0,95**
RWY 25L	1170	1174	0,997	
RWY 02	2283	2292	0,996	
RWY 25R	595	598	0,995	
Taxi	17244	17369	0,993	
Apron	**3106**	**3177**	**0,978**	**0,7**
Apron T1	2648	2707	0,978	
Apron T2	458	470	0,974	
Stands	**29761**	**30015**	**0,992**	**0,5**
Stands T1	15447	15548	0,993	
Stands T2	14314	14467	0,989	
Airborne	**62186**	**68062**	**0,914**	**0,95**

4.3 Resto de vehículos

Aun habiendo filtrado estos vehículos, en otras grabaciones aparecen aun algunos vehículos que transmiten a tasas diferentes a las nominales. Un ejemplo de ello lo encontramos en el fichero grabado en verano de 2019, un vehículo con ICAO Address 34540F y Call Sign “LAB1” el cual recorre de punta a punta varias veces la RWY 25R. Debido a que no se puede tener en cuenta todos estos vehículos (no se dispone de una lista completa de todos ellos), se añade una opción en la aplicación que permite ver que vehículos están transmitiendo de una manera no nominal y te deja añadirlo a la lista de vehículos descartados. También te deja ver sobre el mapa de manera aislada los paquetes transmitidos por ese transponder para determinar si es algún vehículo terrestre que realmente no transmite adecuadamente o si es un avión que si transmite correctamente pero MLAT no lo está captando (quizá por mal funcionamiento de MLAT en una zona determinada…). La idea es poder escoger objetivamente que vehículo descartas porque realmente no transmite adecuadamente y añade ruido en el cálculo de parámetros y cual dejas en la grabación porque realmente no es ruido, sino deficiencias de la MLAT que hay que tener en cuenta para calcular dichos parámetros.

En la figura **(Fig. 4.2)** se puede ver la interfaz que ofrece la aplicación para tratar con los vehículos Squitter. A la izquierda aparecen listados los vehículos ya descartados de la grabación (en verde los que en ese fichero en particular han aparecido y se han filtrado). A la derecha aparecen todos los vehículos que aparecen en el fichero y no han sido descartados. La columna de Update Rate (UR) de la tabla de la derecha permite ver que vehículos transmiten de forma

anormal y pasarlos de la tabla de la derecha a la tabla de la izquierda. Además la tabla de la derecha también te permite ver la diferencia entre los paquetes recibidos de un vehículo y los que se esperaban recibir.

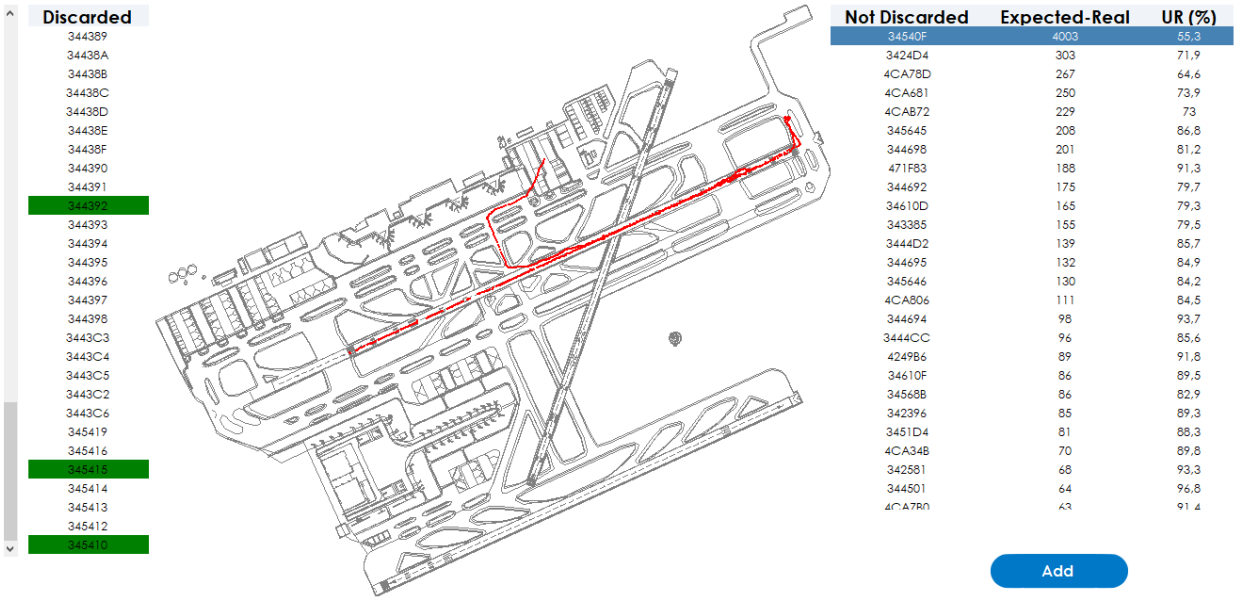

Fig. 4.2 Vehículo con Call Sign LAB1 siendo añadido a la lista de vehículos descartados. Tiene un UR del 55% y 4000 paquete que no se han recibido

Se usa este método porque en el mensaje no hay ningún campo que indique si el paquete se ha enviado o no mediante Squitter. Hay un campo que indica si es “Ground Vehicle” o “Aircraft”, pero no todos los vehículos terrestres transmiten de manera inadecuada, y no hay ninguna razón para filtrarlos todos de la grabación ya que también es importante que MLAT los detecte correctamente.

4.4 Transponders fijos de referencia

4.4.1 Localización

Otro factor a tener en cuentas son los transponders transmisores fijos que hay repartidos por LEBL y que sirven como transponders de referencia para la MLAT. El hecho de que estos transponders aparezcan durante toda la grabación falsea el valor final de los diferentes parámetros, ya que en general la MLAT detecta correctamente estos transponders, favoreciendo los resultados.

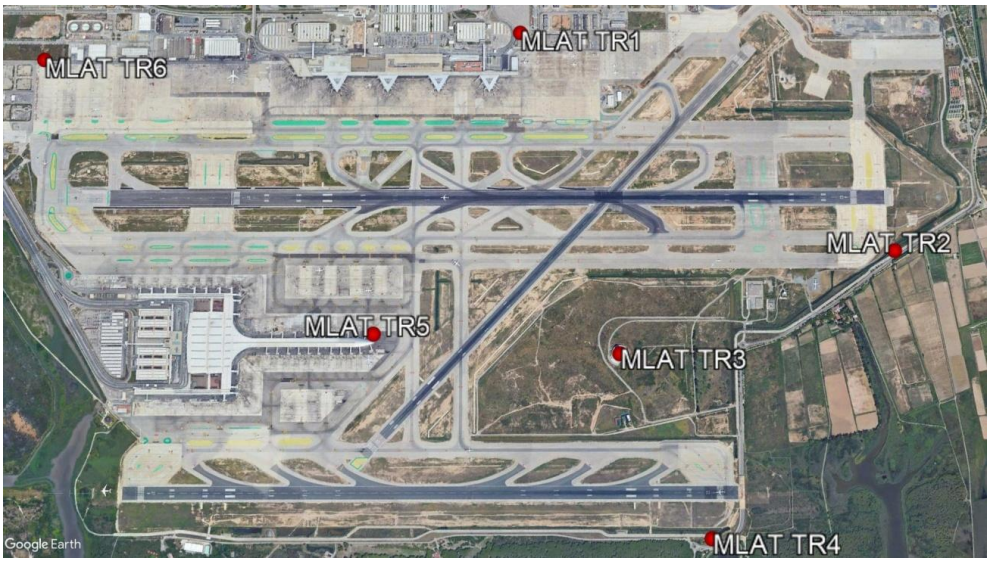

Fig. 4.3 Localización de los transponders de referencia de la MLAT en LEBL

Por ello se eliminan de la grabación. Inicialmente se plantea eliminarlos a partir de su posición conocida, lo que supondría crear un área encima de cada transponder y omitir los paquetes que provengan de esa área. El problema que supone este método es que también eliminaría posibles vehículos que pasaran por esa área.

Fig. 4.4 Omisión de los paquetes recibidos desde las zonas donde se encuentran los transponders de referencia de la MLAT en LEBL

La segunda opción, más simple y que finalmente se aplica, es filtrar los paquetes por ICAO Address, ya que en el caso de los transponders de referencia es conocido. De esta manera se puede aprovechar el código ya desarrollado para eliminar los vehículos que usan Squitter.

Barcelona tiene un total de 6 transponders transmisores que sirven de referencia para la MLAT, los cuales están listados en la tabla **(Tabla 4.4)**.

Tabla 4.4 ICAO Address de los transponders de referencia de la MLAT

Tx	ICAO Address
1	342384
2	342387
3	342386
4	342385
5	342383
6	3433D5

4.4.2 Resultados

En el fichero del recorrido de Barcelona del verano de 2019, de una duración de 4h, se pasa de 153.338 paquetes leídos a 81.126 paquetes leídos.

Puesto que 5 de los 6 transponders se encuentran en la zona de Airborne, ahí es dónde se notan las diferencias principales, sobre todo en el Update Rate, dónde pasa de un 94,18% a un **77,58%**.

CAPITULO 5. Resultados evaluación MLAT LEBL

5.1 Comparativa entre trafico de oportunidad y D-GPS

El fichero analizado para determinar los resultados del funcionamiento del sistema MLAT será el de verano de 2019. La tabla **(Tabla 5.1)** muestra los resultados que ofrece la aplicación con tráfico de oportunidad. Por otro lado la tabla **(Tabla 5.2)** muestra los resultados que ofrece la aplicación analizando solo el vehículo equipado con D-GPS.

Tabla 5.1 Resultados generales del análisis de tráfico de oportunidad. Resultados en porcentaje (%), excepto PA (m)

	Maneouvering Area	Apron	Stands	Airborne
UR	99.55	98.30	99.61	77.58
PA	-	-	-	-
Probability of MLAT Detection	99.91		99.91	-
PID	100			
PFD	0.011			
PFI	0			

Tabla 5.2 Resultados generales del análisis del vehículo equipado con D-GPS. Resultados en porcentaje (%), excepto PA (m)

	Maneouvering Area		Apron		Stands		Airborne	
UR	99.22		97.64		-		99.80	
PA (P95 y P99)	7.48	11.82	8.26	10.26	-	-	-	-
Probability of MLAT Detection	99.90				-		-	
PID	100							
PFD	0							
PFI	0							

El sistema MLAT cumple de manera global con las especificaciones del doc. ED-117. Solo hay algún parámetro que supera ligeramente el límite en alguna zona determinada.

Para el cálculo con tráfico de oportunidad, el UR en la zona de Airborne da unos resultados muy bajos. Esto es así porque se evalúan todos los datos de Airborne que capta la MLAT, cuando en realidad está diseñada para funcionar

correctamente en la zona aeroportuaria y en sus inmediaciones. Si se descarta todo el Airborne y solo se analizan las zonas de tipo 4 y 5 los resultados si cumplen con las especificaciones (ofreciendo un UR de 99.15% y 98.31% respectivamente).

En ningún fichero que se haya analizado, con ninguno de los dos métodos, se ha encontrado una PID inferior al 100%. Tampoco se ha encontrado nunca una PFI superior al 0%, lo cual es consecuencia de la similitud entre ambos conceptos y del método en que se calculan.

La PFD con tráfico de oportunidad también está por encima del umbral. Aun así la aplicación solo detecta falsas detecciones en las zonas de tipo 4 y 5, donde se ha utilizado un método aproximativo para calcularlo. Cuando se calcula con los datos de D-GPS da una PFD del 0%, aunque no se pueden comparar los datos de las áreas de tipo 4 y 5 debido a que D-GPS no ofrece datos de esas zonas. A parte, el doc. ED-117 no especifica que se tengan que evaluar las áreas de tipo 4 y 5 para este parámetro, sino que se trata de una opción que se ha decidido incorporar en la apkicación.

5.2 Resultados de Position Accuracy

Mirando la tabla **(Tabla 5.2)** se observa que la PA también se mantiene debajo de los límites, exceptuando el P95 en la zona de Apron, dónde está 0.76 m por encima del límite. Esto se puede deber al apantallamiento que se produjo al entrar en esa zona, debido a que se pasó cerca de un gran número de aeronaves estacionadas en los stands remotos de la T1 norte. En la figura **(Fig. 5.1)** se puede observar el incremento del error en posición seguido de la pérdida de la señal MLAT que se produce al entrar en el Apron de la T1.

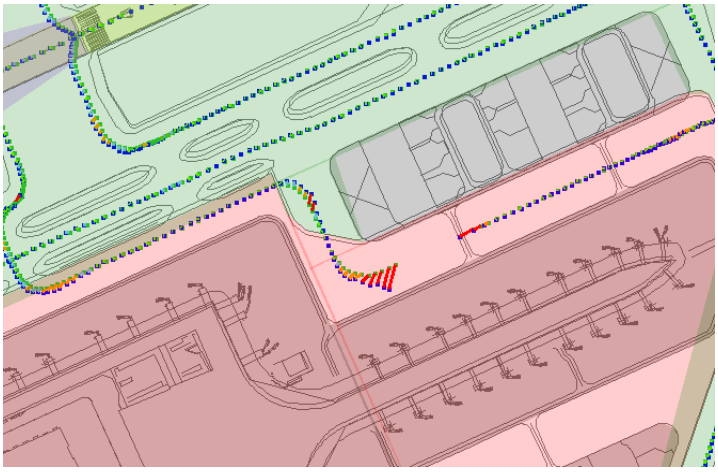

Fig. 5.1 Apantallamiento en la zona Apron de la T1

Analizando los gráficos Scatter de la zona de maniobras **(Fig. 5.2)** y de la zona de Apron **(Fig. 5.3)** se observa que la mayoría de los datos se encuentran dentro del límite de 12 m. Solo algunos puntos tienen un error en posición mayor, el cual se debe en parte al apantallamiento que se produjo en el Apron de la T1 **(Fig. 5.1)**; aunque principalmente se produce en las zonas de giros, sobretodo en el By-pass que une la T1 con la T2 **(Fig 5.4)** y en la cabecera de la pista 07R **(Fig. 5.5)**.

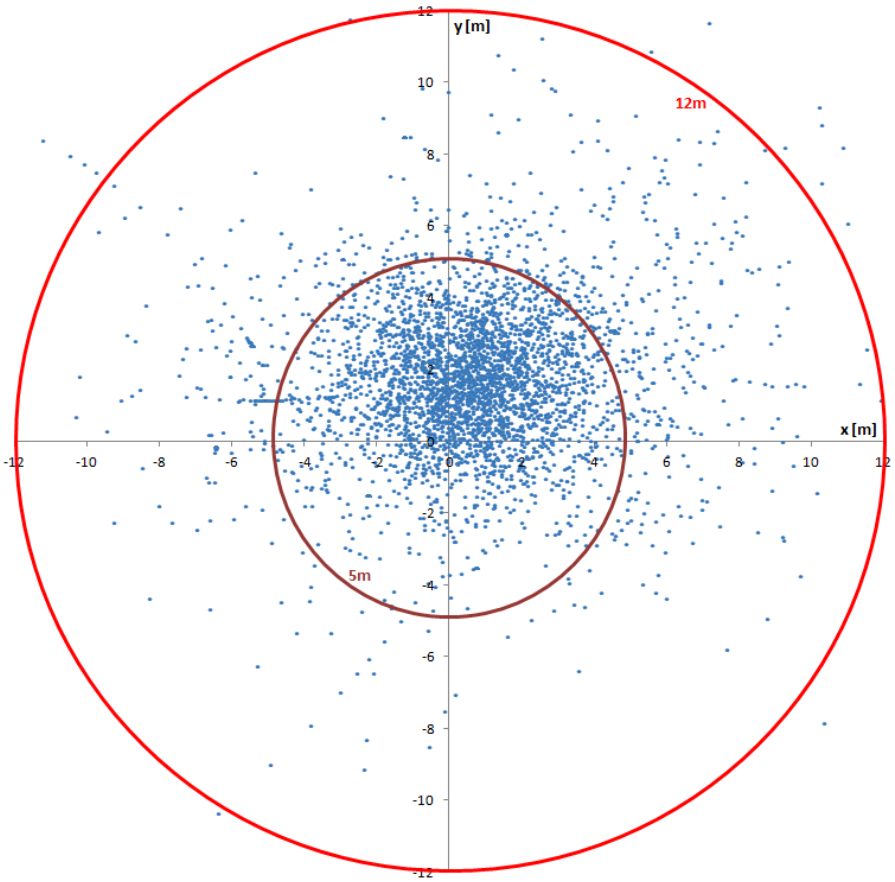

Fig. 5.2 Scatter zona de maniobras (las 3 pistas y taxi)

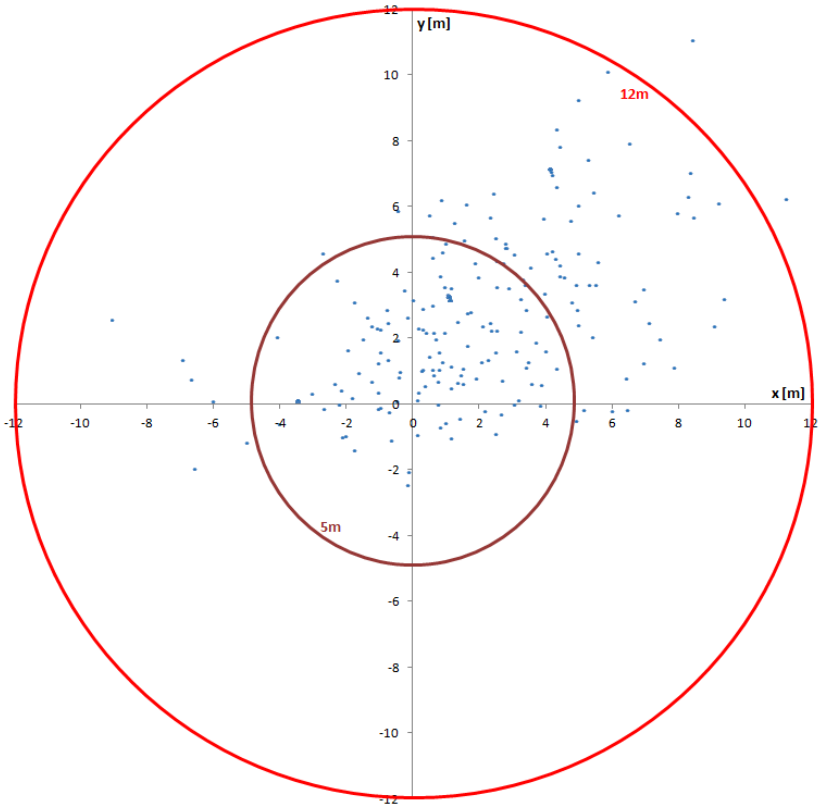

Fig. 5.3 Scatter Apron (solo datos de la T1)

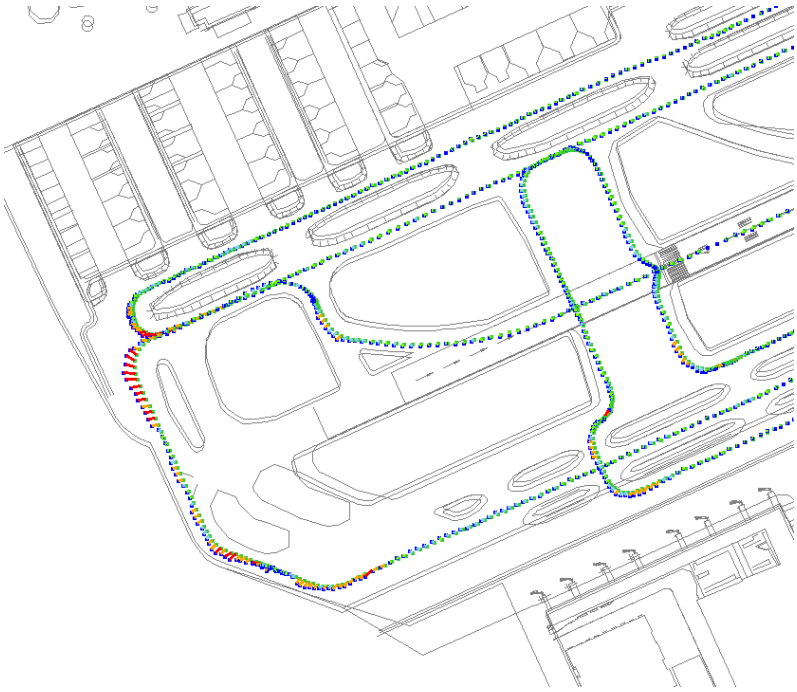

Fig. 5.4 By-pass T1-T2

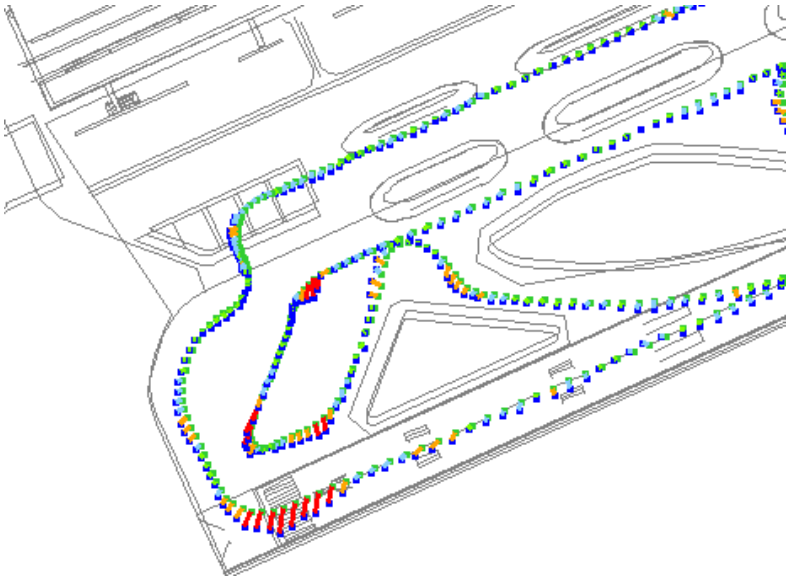

Fig. 5.5 Cabecera de la pista 07R

La aplicación muestra los puntos D-GPS que han sido emparejados con un punto de la MLAT, es por esto que en la **Fig. 5.4** no aparece la trayectoria recibida por el D-GPS en el tramo dónde no se reciben datos de la MLAT por apantallamiento.

El motivo por el cual se produce un error mayor en posición en la zona de la cabecera de la pista 07R **(Fig. 5.5)** es que una de las estaciones MLAT que se encuentra en esa área de influencia está apagada y fuera de servicio.

CONCLUSIONES

El objetivo del trabajo era crear una aplicación funcional que evaluará los parámetros de actuación del sistema de multilateración de Barcelona. Se tenía que poder obtener resultado para todos los parámetros partiendo solo de tráfico de oportunidad, lo cual permitiría evaluar periódicamente en intervalos cortos la MLAT debido a que se simplificaría mucho la metodología y operativa respecto al método usado normalmente (vehículo terrestre equipado con D-GPS).

El resultado ha sido que la mayoría de parámetros se han podido evaluar con tráfico de oportunidad, a excepción del parámetro de position accuracy, el cual se ha tenido que calcular con datos obtenidos del D-GPS. Puesto que con una futura actualización del sistema MLAT se podría calcular este parámetro con tráfico de oportunidad, lo cual da continuidad al trabajo, se considera que en global se han conseguido los objetivos del trabajo.

Asimismo se ha conseguido que la aplicación se diferencie de las aplicaciones que existían actualmente para calcular estos parámetros.

Por otro lado también se estudió la posibilidad de ampliar las capacidades de la aplicación pudiendo evaluar también el sistema SMR de Barcelona. Está opción se descartó al hacer un estudio de la información que llegaba en los mensajes de este sistema, la cual era muy escasa para poder crear un algoritmo que pudiera analizar estos datos de forma genérica, debido a la complejidad. Aún así no se descarta esta opción en una posible continuación del trabajo.

Respecto al funcionamiento de la MLAT, se determina que funcionamientos es correcto en su cómputo global, con algunas excepciones que podrían ser mejoradas para acabar de cumplir con las especificaciones del MOPS de EUROCAE.

BIBLIOGRAFÍA

[1] EUROCAE, Minimum operational performance specification for surface movement radar sensor systems for use in advanced surface movement guidance and control systems (A-SMGCS) (ED-116), 2004.

[2] EUROCAE, Minimum operational performance specification for mode S multilateration systems for use in advanced surface movement guidance and control systems (A-SMGCS) (ED-117), 2003.

[3] EUROCONTROL, Specification for Surveillance Data Exchange - Part 1, 2016.

[4] EUROCONTROL, Standard document for surveillance data exchange part 7: category 010 transmission of Monosensor Surface Movement Data, 2007.

[5] EUROCONTROL, Specification for Surveillance Data Exchange ASTERIX Part 14 Category 20 Multilateration Target Reports, 2015.

[6] EUROCONTROL, Standard document for surveillance data exchange part 12: category 021 ADS-B messages, 2003.

[7] EUROCONTROL, Specification for Surveillance Data Exchange ASTERIX Part 14 Category 020 Multilateration Target Reports Appendix A: Reserved Expansion Field, 2016.

[8] Aena, dirección de servicios, división de navegación y vigilancia departamento radar, Informe de verificación técnica del SMMS del aeropuerto de Barcelona, 2012.

[9] EUROCAE, Minimum Operational Performance Standards for 1090 MHz Extended Squitter Automatic Dependent Surveillance – Broadcast (ADS-B) and Traffic Information Services – Broadcast (TIS-B), 2012.

[10] S. P. Drake, "*Converting GPS Coordinates (ϕλh) to Navigation Coordinates (ENU)*", DSTO Electronics and Surveillance Research Laboratory, 2002.

[11] Spreadsheetlight, Librería open source para .NET Framework compatible con Microsoft Excel < https://spreadsheetlight.com/ >, funcional a 6 de septiembre de 2019.

ANEXOS

Código de la aplicación: ED-MLAT Performance Evaluator

FORMS

Program.cs

```
using System;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
   static class Program
   {
      /// <summary>
      /// Punto de entrada principal para la aplicación.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Form1 form1 = new Form1();
         form1.FormClosed += AllForms_Closed;
         form1.Show();
         Application.Run();
      }

      static bool NoCerrar = false;

      private static void AllForms_Closed(object sender, FormClosedEventArgs
e) //La aplicacion se cierra cuando todas las ventanas se cierran
      {
         ((Form)sender).FormClosed -= AllForms_Closed;

         if (Application.OpenForms.Count == 0)
         {
            if (!NoCerrar)
            {
               Application.ExitThread();
            }
            else
            {
               NoCerrar = false;
               Form1 form1 = new Form1();
               form1.FormClosed += AllForms_Closed;
               form1.SetAppInterrumpida(true);
               form1.Show();
```

```
            }
        }
        else
        {
            Application.OpenForms[0].FormClosed += AllForms_Closed;
        }
    }

    public static void SetNoCerrar(bool loading)
    {
        NoCerrar = loading;
    }
  }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using Codigo;

namespace ED_SMR_MLAT_Performance
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      LectorMensaje myList = new LectorMensaje();
      LectorMensaje myListDGPS = new LectorMensaje();
      LectorMensaje myListConDescartados = new LectorMensaje();
      string nombreFichero = "";//Guarda el nombre del fichero .ast leido
      bool nuevoFicheroMain = false; //Controla si se genera la form desde
escritorio o desde Main Form
      bool cerrarMain = false;//boleano que cierra el Main si se carga bien el
fichero (desde el main)
      Point labelInfoLoc; //Guarda la posicion incial de la label informativa (Label
info de fichero despues de load)
      bool appInterrupida = false; //A veces la aplicacion se cierra sola, este
boleano controla esa interrupción y la impide.

      private void Form1_Load(object sender, EventArgs e)
      {
         labelInfoLoc = labelInformativa.Location;
         if (!nuevoFicheroMain)
         {
            labelWelcome.Text = "Welcome to ED-MLAT Performance\n
Evaluator";
            labelSubWelcome.Text = "This application calculates the value of the
different parameters\nlisted in the document of EUROCAE “MOPS for Mode S
MLAT\nSystems” (ED-117) using .ast files of opportunity traffic from
LEBL,\nand determines if the MLAT meets these minimum requirements.\nIt
also allows analyzing D-GPS recordings to calculate position\naccuracy data;
and it provides an interface that allows you to\ndiscard certain vehicles from the
analysis.";
            if (appInterrupida)
            {
               appInterrupida = false;
```

```
                CrearFormInformativa("An error has occurred. Something", "went
wrong. Please try again.", 347, 116, 27, 46);
            }
        }
        else
        {
            LoadFileButton.PerformClick();
        }
    }

    private void LoadFileButton_Click(object sender, EventArgs e)
    {
        string nombre;
        try
        {
            this.openFileDialog1.FileName = "";
            this.openFileDialog1.Filter = "File documents (.ast)|*.ast";
            Program.SetNoCerrar(true);
            this.openFileDialog1.ShowDialog();
            Program.SetNoCerrar(false);
            if (this.openFileDialog1.FileName.Equals("") == false)
            {
                nombre = (this.openFileDialog1.FileName);
                Cursor.Current = Cursors.WaitCursor;
                LectorMensaje listTemporal = new LectorMensaje();//Si ya hemos
cargado una lista, no queremos que se sobre escriba con una list que no se
cargue correctamente
                Program.SetNoCerrar(true);
                listTemporal.CargarListaDeDirectorio(nombre);
                Program.SetNoCerrar(false);
                if (listTemporal.GetNumList() != 0)
                {
                    if (!listTemporal.TodoSMR())
                    {
                        Program.SetNoCerrar(true);
                        if (nuevoFicheroMain)
                        {
                            cerrarMain = true;
                            nuevoFicheroMain = false;
                        }
                        myListConDescartados = listTemporal;

                        bool diferentesSIC =
myListConDescartados.ComprobarDiferentesSIC();//Controla si hay SMR y
MLAT en un mismo fichero
                        if (diferentesSIC)
                        {
                            CrearFormInformativa("This file contains both SMR and
MLAT packages. Only MLAT", "can be evaluated. SMR packages will be
discarded.", 518, 201, 17, 41);
```

```
                    Cursor.Current = Cursors.WaitCursor;
                    LectorMensaje listaTemporal = new LectorMensaje();
                    listaTemporal =
myListConDescartados.SepararSMRyADSB();
                    myListConDescartados.ClearList();
                    myListConDescartados = listaTemporal;
                }

myListConDescartados.OrdenarPaquetesPorTiempo();//Corrige algunos
paquetes no ordenados
                myListConDescartados.SetUTCcorregido(); //Coregimos la
hora para cambio de dia. Se hace despues de separar MLAT y SMR, ya que
                                                    //cuando estan juntos los ficheros no
estan perfectamente ordenados en tiempo y la funcion no funciona
                myList =
myListConDescartados.DescartarVehiculosSquitter();//Tambien ordenamos en
tiempo los vehiculos Squitter, puesto que despues se pueden
                                                                //llegar a usar en
MainForm

                nombreFichero = this.openFileDialog1.SafeFileName;

                labelWelcome.Visible = false;
                labelSubWelcome.Visible = false;
                ResultadosLoad();
                AvaluateButton.Visible = true;
                AddDGPSButton.BackColor = Color.FromArgb(255, 128, 0);
                AddDGPSButton.FlatAppearance.MouseOverBackColor =
Color.FromArgb(255, 128, 0);
                myListDGPS.ClearList();
                AddDGPSButton.Visible = true;
                LoadFileButton.Visible = false;
                Program.SetNoCerrar(false);
            }
            else
            {
                CrearFormInformativa("The file only contains SMR data. This
program only evaluates MLAT.", "The program is also able to filter MLAT from
(SMR + MLAT) files.", 575, 225, 22, 42);
                if (nuevoFicheroMain)
                {
                    this.Close();
                }
            }
        }
        else
        {
            CrearFormInformativa("Error while loading the file.", "Maybe
wrong .ast category?", 280, 80, 22, 12);//Valores puestos a mano para
cuadradrar el texto segun la longitud del propio texto
```

```
                    if (nuevoFicheroMain)
                    {
                        this.Close();
                    }
                }
            }
            else
            {
                CrearFormInformativa("Please, ", "select a file first.", 173, 28, 46,
15);
                if (nuevoFicheroMain)
                {
                    this.Close();
                }
            }
        }
        catch (FormatException)
        {
            CrearFormInformativa("Error while loading the file,", "incorrect data
structure.", 261, 73, 17, 18);
            return;
        }
        catch (FileNotFoundException)
        {
            CrearFormInformativa("File not", "found.", 155, 19, 40, 43);
            return;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    private void AddDGPSButton_Click(object sender, EventArgs e)
    {
        string nombre;
        string icao = "";
        ICAOcode address = new ICAOcode();
        address.ShowDialog();
        icao = address.ExportarIcao();
        if (icao != " ")
        {
            try
            {
                List<string> descartados =
myListConDescartados.LeerVehiculosSquitter();
                int i = 0;
                bool encontrado = false;
                bool respuesta = false;
```

```
            while ((i < descartados.Count()) && (!encontrado))
            {
                if (icao == descartados[i])
                {
                    encontrado = true;
                    respuesta = CrearFormInformativaYesNo("This ICAO Address
is in the list of discarded vehicles.", "Do you want to remove it from the list?",
476, 97, 241, 27, 71); //Si el ICAO Address D-GPS esta en la lista de
descartados
                    if (respuesta)
                    {
                        Cursor.Current = Cursors.WaitCursor;
                        descartados.Remove(icao);
                        File.Delete("VehiculosAEliminar.txt");
                        File.WriteAllLines("VehiculosAEliminar.txt", descartados);
                        myList.ClearList();
                        myList =
myListConDescartados.DescartarVehiculosSquitter();
                        ResultadosLoad();
                    }
                }
                i++;
            }
            if ((encontrado && respuesta) || (!encontrado))//Si esta en la lista
de descartados y se elimina, o sino esta en la lista
            {
                this.openFileDialog1.FileName = "";
                this.openFileDialog1.Filter = "File documents (.txt)|*.txt";
                this.openFileDialog1.ShowDialog();
                if (this.openFileDialog1.FileName.Equals("") == false)
                {
                    nombre = (this.openFileDialog1.FileName);
                    Cursor.Current = Cursors.WaitCursor;
                    LectorMensaje listTemporal = new LectorMensaje();//Si ya
hemos cargado una lista, no queremos que se sobre escriba con una list que
no se cargue correctamente
                    listTemporal.CargarListaDeDirectorioDGPS(nombre, icao);
                    if (listTemporal.GetNumList() != 0)
                    {
                        int matches =
listTemporal.SetIndicesDGPS(myList);//Devuelve matches y hace
SetIndiceDGPS()
                        List<int> a = new List<int>();
                        for (int j = 0; j < listTemporal.GetNumList(); j++)
                        {
                            if (listTemporal.GetPlanI(j).GetIndiceDGPS() != -1)
                            {
                                a.Add(j);
                            }
                        }
```

```
                    Matches form = new Matches();
                    form.ImportarMatches(matches);
                    form.ShowDialog();
                    if (matches != 0)
                    {
                        myListDGPS = listTemporal;
                        AddDGPSButton.BackColor = Color.LimeGreen;
                        AddDGPSButton.FlatAppearance.MouseOverBackColor
= Color.LimeGreen;
                    }
                }
                else
                {
                    CrearFormInformativa("Error while loading the file.", "Maybe
wrong format?", 261, 73, 17, 18);
                }
            }
            else
            {
                CrearFormInformativa("Please, ", "select a file first.", 173, 28,
46, 15);
            }
        }
        catch (FormatException)
        {
            CrearFormInformativa("Error while loading the file,", "incorrect data
structure.", 261, 73, 17, 18);
            return;
        }
        catch (System.IndexOutOfRangeException)
        {
            CrearFormInformativa("Error while loading the file,", "incorrect data
structure.", 261, 73, 17, 18);
            return;
        }
        catch (FileNotFoundException)
        {
            CrearFormInformativa("File not", "found.", 155, 19, 40, 43);
            return;
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }
}

private void AvaluateButton_Click(object sender, EventArgs e)
```

```
        {
            try
            {
                bool respuesta = true;
                if (myListDGPS.GetNumList() == 0)
                {
                    respuesta = CrearFormInformativaYesNo("No D-GPS file has been
added, so some features of the application", "won't be available (mainly position
accuracy results).", 581, 166, 310, 27, 70);
                }
                if (respuesta)
                {
                    MainForm evaluar = new MainForm();
                    evaluar.SetMensaje(myList, myListDGPS, myListConDescartados);
                    string nuevoNombreFichero = "";
                    if (nombreFichero.Length >= 4)
                    {
                        for (int n = 0; n < nombreFichero.Length - 4; n++)//Recortamos el
".ast"
                            nuevoNombreFichero = nuevoNombreFichero +
nombreFichero[n];
                    }
                    evaluar.SetNombreFichero(nuevoNombreFichero);
                    evaluar.StartPosition = FormStartPosition.CenterScreen;
                    evaluar.WindowState = FormWindowState.Maximized;
                    evaluar.Text = nuevoNombreFichero + " - ED-MLAT Permormance
Evaluator";
                    evaluar.Show();
                    this.Close();
                }
            }
            catch (FormatException)
            {
                CrearFormInformativa("Incorrect data", "structure.", 193, 36, 27, 46);
                return;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return;
            }
        }

        public void SetNuevoFichero(bool nuevo)
        {
            this.nuevoFicheroMain = nuevo;
        }

        public void SetAppInterrumpida(bool interrumpido)
        {
```

```
            this.appInterrupida = interrumpido;
        }

        public void ResultadosLoad() //establece el valor de las label de info
despues de cargar un .ast
        {
            string nombreInformativo = this.openFileDialog1.SafeFileName;
            string nombreInformativo1 = "";
            string nombreInformativo2 = "";
            if (nombreInformativo.Count() > 34)//Controla que el titulo no sea
demasiado largo y salga de la form
            {
                for (int i = 0; i < 34; i++)
                {
                    nombreInformativo1 = nombreInformativo1 +
Convert.ToString(nombreInformativo[i]);
                }
                for (int i = 34; i < nombreInformativo.Count(); i++)
                {
                    nombreInformativo2 = nombreInformativo2 +
Convert.ToString(nombreInformativo[i]);
                }
            }
            string avionesDetectados = "";
            int aircraftDetected = myList.AircraftDetected();
            if (aircraftDetected != 0)
                avionesDetectados = "\n          from " + aircraftDetected +
                " different transponders";
            if (nombreInformativo1 != "")
            {
                labelInformativa.Text = nombreInformativo1 + "\n" +
nombreInformativo2 + " correctly\nloaded\n\n" +
                    myList.GetNumList() + " packages readed" + avionesDetectados +
"\n\nbetween " + myList.GetPlanI(0).ConvertUTC("SinDecimas") +
                    " hours\n        and " + myList.GetPlanI(myList.GetNumList() -
1).ConvertUTC("SinDecimas") + " hours";
                labelInformativa.Location = new Point(labelInfoLoc.X - 20,
labelInfoLoc.Y - 20);
            }
            else
            {
                labelInformativa.Text = this.openFileDialog1.SafeFileName + "
correctly\nloaded\n\n" +
                    myList.GetNumList() + " packages readed" + avionesDetectados +
"\n\nbetween " + myList.GetPlanI(0).ConvertUTC("SinDecimas") +
                    " hours\n        and " + myList.GetPlanI(myList.GetNumList() -
1).ConvertUTC("SinDecimas") + " hours";
                labelInformativa.Location = new Point(labelInfoLoc.X, labelInfoLoc.Y);
            }
        }
```

```
        public void CrearFormInformativa(string info1, string info2, int formWidth,
int Xbutton, int Xlabel1, int Xlabel2)//Valores puestos a mano para cuadradrar el
texto segun la longitud del propio texto
        {
            FormInformativa formI = new FormInformativa();
            formI.ImportarInformacion(info1, info2, formWidth, Xbutton, Xlabel1,
Xlabel2);
            formI.ShowDialog();
        }

        public bool CrearFormInformativaYesNo(string info1, string info2, int
formWidth, int Xbutton1, int Xbutton2, int Xlabel1, int Xlabel2)//Valores puestos
a mano para cuadradrar el texto segun la longitud del propio texto
        {
            FormInformativaYesNo formI = new FormInformativaYesNo();
            formI.ImportarInformacion(info1, info2, formWidth, Xbutton1, Xbutton2,
Xlabel1, Xlabel2);
            formI.ShowDialog();
            return (formI.ExportarRespuesta());
        }

        public bool CerrarMain()//Si se carga fichero desde el main, y se carga
correctamente, cierra el main
        {
            return cerrarMain;
        }

        private void closeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }
    }
                                                }
```

MainForm.cs

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using Codigo;
using System.IO;
using SpreadsheetLight;

namespace ED_SMR_MLAT_Performance
{
   public partial class MainForm : Form
   {
      public MainForm()
      {
         InitializeComponent();
      }

      LectorMensaje myList;
      LectorMensaje myListDGPS;
      LectorMensaje myListConDescartados;
      CListaMap myLmap;
      Bitmap myBitmap;
      Bitmap myBitmapDescartados;
      Avaluador myAvaluador;
      Avaluador myAvaluadorDGPS;
      List<List<int>> aircraftDetected;//Matriz con los aviones estructurados
segun su ICAO Adress
      string nombreFichero = "";//nombre del .ast que se esta leyendo
      bool saveAllCorrecto = false; //Controla si se ha guardado todo
correctamente
      bool[] saveAllCorrectoPorSeparado = new bool[2];//Controla si se ha
guardado la foto y todos los excels, por lo que ya no haria falta clicar en
SaveAll "general"
      bool segmentation = false;//Controla si se ha clicado en segmentation
      bool zoom = false; //En el mapa esta false cuando esta en modo zoom in
(predeterminado)
      bool HideBack = false; //En el mapa controla si se esconde o no el mapa.
False si se muestra el mapa
      bool MLATandDGPS = false; //En el mapa controla si se quiere ver o no
DGPS + MLAT
      bool DGPS = false; //En el mapa controla si se quiere ver o no DGPS
      int grosor = 1; //Grosor del trazo en SaveBitmap
      bool DGPSresults = false; //False cuando se ve el summary de MLAT, true
con summary D-GPS
      bool mapaDescartados = false; //Si estamos viendo el mapa de
descartados no hay opciones de zoom, segmentacion, etc.
```

```
        double[,] dimensionesDescartado; //Cambia la dimensiones del mapa de
descartados en caso de que clique "View Whole"
        double proporcionDescartados; //Cambia la proporcion del mapa de
descartados en caso de que clique "View Whole"

        private void MainForm_Load(object sender, EventArgs e)
        {
            Cursor.Current = Cursors.WaitCursor;
            panelContenedorMapa.Visible = false;
            panelContenedorResultados.Visible = true;
            panelContenedorDescartados.Visible = false;
            saveAllCorrecto = false;
            myAvaluador = new Avaluador(myList);
            aircraftDetected = myList.AircraftICAODetected();
            myLmap = new CListaMap();
            if (myListDGPS.GetNumList()!=0)//Si no hay DGPS se ocultan algunas
prestaciones
            {
                myAvaluadorDGPS = new Avaluador(myList, myListDGPS);
            }
            else
            {
                DGPSresultsButton.Visible = false;
                SaveScatterButton.Visible = false;
                ButtonDGPS.Visible = false;
                ButtonMLAT_DGPS.Visible = false;
            }
            InicializarBotones();
            InicializarTablas();
            InicializarMapas();
        }

        public void RecargarMainForm()//Recarga los valores de la form despues
de añadir / descartar un vehiculo
        {
            Cursor.Current = Cursors.WaitCursor;
            myList.ClearList();
            myList = myListConDescartados.DescartarVehiculosSquitter();
            if (myListDGPS.GetNumList() != 0)
            {
                myListDGPS.SetIndicesDGPS(myList);
            }

            saveAllCorrecto = false;
            saveAllCorrectoPorSeparado = new bool[2];
            myAvaluador = new Avaluador(myList);
            aircraftDetected = myList.AircraftICAODetected();
            if (myListDGPS.GetNumList() != 0)
            {
                myAvaluadorDGPS = new Avaluador(myList, myListDGPS);
```

```
            }
            string icao="";
            string textoBoton="";
            if (AddRemoveDiscardButton.Text=="Add")
            {
                icao = Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.CurrentRow.Index].Value);
                textoBoton = "added";
            }
            else if(AddRemoveDiscardButton.Text == "Remove")
            {
                icao = Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.CurrentRow.Index].Value);
                textoBoton = "removed";
            }
            InicializarBotones();
            InicializarTablas();
            InicializarMapas();
            panelMapDescartados.Invalidate();

            FormInformativa formI = new FormInformativa();
            formI.ImportarInformacion("Vehicle with ICAO Address", icao + "
correctly " + textoBoton + ".", 276, 74, 20, 24);
            formI.ShowDialog();
        }

        public void ResetearBoleanos()//Pone los boleanos en su estado original
        {
            MLATandDGPS = false;
            DGPS = false;
            segmentation = false;
            zoom = false;
            HideBack = false;
            DGPSresults = false;
        }

        public void SetMensaje(LectorMensaje lista, LectorMensaje listaDGPS,
LectorMensaje listaSinDescartados)
        {
            this.myList = lista; //Carga la lista MLAT de la Form1 a MainForm
            this.myListDGPS = listaDGPS; //Carga la lista DGPS de la Form1 a
MainForm
            this.myListConDescartados = listaSinDescartados; //Carga la lista MLAT
sin vehiculos descartados de la Form1 a MainForm
        }

        public void SetNombreFichero(string nombre)//nombre .ast que se esta
leyendo
        {
            this.nombreFichero = nombre;
```

```
        }

        public void InicializarBotones()//Pone el texto y booleanos en el estado
inicial
        {
            buttonHideBackground.Text = "Hide Background";
            buttonViewSegmentation.Text = "View Segmentation";
            ButtonZoom.Text = "Zoom Out";
            ButtonDGPS.Text = "View only D-GPS";
            segmentation = false;//Controla si se ha clicado en segmentation
            zoom = false; //En el mapa esta false cuando esta en modo zoom in
(predeterminado)
            MLATandDGPS = false; //En el mapa controla si se quiere ver o no
DGPS mas MLAT
            DGPS = false; //En el mapa controla si se quiere ver o no DGPS
            HideBack = false;
            grosor = 1; //Grosor del trazo en SaveBitmap
            DGPSresultsButton.Text = "View D-GPS Results";
            DGPSresults = false; //False cuando se ve el summary de MLAT, true
con summary D-GPS
            AddRemoveDiscardButton.Visible = false;
            ViewWholeButton.Visible = false;
        }

        public void InicializarTablas()//Crea el contenido de las tablas
        {
            CrearTablaParameters();
            CrearTablaSummary(this.SummaryGrid, myAvaluador);
            CrearTablaDescartados(this.tablaGridDescartados);
            tablaGridDescartados.ClearSelection();
            CrearTablaNoDescartados(this.tablaGridNoDescartados);
            tablaGridNoDescartados.ClearSelection();
        }

        public void InicializarMapas()//Crea el contenido de los mapas
        {
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Genera el bmp con el mapa de LEBL
            PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height); //Pinta
los paquetes sobre el mapa
            myBitmapDescartados = new Bitmap(this.panelMapDescartados.Width,
this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            dimensionesDescartado = myLmap.GetDIMENSIONES();
            proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
```

```
            PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height); //Genera
el bmp con el mapa de LEBL
        }

        private void panelMap_Paint(object sender, PaintEventArgs e)
        {
            try
            {
                Graphics graphics = e.Graphics;
                graphics.DrawImage(myBitmap, 0, 0, myBitmap.Width,
myBitmap.Height);
                graphics.Dispose();
            }
            catch (Exception)
            {
                return;
            }
        }

        private void PintarFondoPantalla(Bitmap Bit, int width, int height)  //Dibuja
el fondo de pantalla con el mapa y lo guarda en bmp
        {
            Graphics graphicsObj;
            graphicsObj = Graphics.FromImage(Bit);
            graphicsObj.Clear(Color.White); // Set Bitmap background color. Black
by default
            Pen myPen = new Pen(Color.Gray, 1 / 2);
            if (mapaDescartados)
            {
                PintarMapa(graphicsObj, myPen, width, height);
            }
            else if (!HideBack)
            {
                PintarMapa(graphicsObj, myPen, width, height);
            }
            graphicsObj.Dispose();
        }

        private void PintarMapa(Graphics graphics, Pen myPen, int width, int
height)//Pinta el mapa de LEBL, se usa dentro de PintarFondoPantalla
        {
            double[,] dimensiones = myLmap.GetDIMENSIONES();
            double proporcion;

            if (mapaDescartados)
            {
                dimensiones = dimensionesDescartado;
                proporcion = proporcionDescartados;
            }
```

```
        else if (zoom)
        {
            dimensiones[0, 0] = dimensiones[0, 0] - 10000;
            dimensiones[1, 0] = dimensiones[1, 0] - 7000;
            proporcion= myLmap.GetPROPORCION(height / 5, width / 5);
        }
        else
        {
            proporcion = myLmap.GetPROPORCION(height, width);
        }

        try
        {
            int indiceFichero = 0;
            while (indiceFichero < myLmap.GetNumList())//Pinta el mapa con
lineas y polilineas
            {
                int i = 0;
                Point[] lineVector = new Point[2];
                while (i <
myLmap.GetPlanI(indiceFichero).GetLinea().GetLength(0))
                {
                    lineVector[0] = new
Point(Convert.ToInt32((myLmap.GetPlanI(indiceFichero).GetLinea()[i, 0] -
dimensiones[0, 0]) / proporcion), Convert.ToInt32(((-
(myLmap.GetPlanI(indiceFichero).GetLinea()[i, 1] - dimensiones[1, 0]) /
proporcion)) + height));
                    //(x - dimensionX) / proporcion; (-(y - dimensionY) / proporcion) +
panelAe.Size.Height
                    lineVector[1] = new
Point(Convert.ToInt32((myLmap.GetPlanI(indiceFichero).GetLinea()[i, 2] -
dimensiones[0, 0]) / proporcion), Convert.ToInt32(((-
(myLmap.GetPlanI(indiceFichero).GetLinea()[i, 3] - dimensiones[1, 0]) /
proporcion)) + height));
                    graphics.DrawPolygon(myPen, lineVector); //Dibujamos la linea
del vector
                    i = i + 1;
                }

                i = 0;
                int longPoli = 0;
                int iPoli = 0;
                bool vectorNOAcabado = true;
                while (i < myLmap.GetPlanI(indiceFichero).GetPoli().GetLength(0))
                {
                    while ((vectorNOAcabado) &&
(myLmap.GetPlanI(indiceFichero).GetPoli()[i, 0] != Math.Pow(10, 8)))
                    {
                        longPoli = longPoli + 1;
                        i = i + 1;
```

```
                    if (i ==
myLmap.GetPlanI(indiceFichero).GetPoli().GetLength(0))
                    {
                        vectorNOAcabado = false;
                        i = i - 1;
                    }
                }
                if (vectorNOAcabado)
                {
                    i = i - longPoli;
                }
                else i = i - longPoli + 1;

                Point[] poliVector = new Point[longPoli];
                while (iPoli < longPoli)
                {
                    poliVector[iPoli] = new
Point(Convert.ToInt32((myLmap.GetPlanI(indiceFichero).GetPoli()[i, 0] -
dimensiones[0, 0]) / proporcion), Convert.ToInt32((-
(myLmap.GetPlanI(indiceFichero).GetPoli()[i, 1] - dimensiones[1, 0]) /
proporcion) + height));
                    iPoli = iPoli + 1;
                    i = i + 1;
                }
                i = i + 1;
                iPoli = 0;
                longPoli = 0;
                graphics.DrawLines(myPen, poliVector); //Dibujamos la linea del
vector
            }
            indiceFichero = indiceFichero + 1;
        }
        myPen.Dispose();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

    private void PintarAll(Bitmap Bit, int width, int height)//Pinta todos los
paquetes de myList
    {
        Graphics graphics;
        graphics = Graphics.FromImage(Bit);
        double[,] dimensiones = myLmap.GetDIMENSIONES();
        double proporcion;

        if (zoom)
```

```
        {
            dimensiones[0, 0] = dimensiones[0, 0] - 10000;
            dimensiones[1, 0] = dimensiones[1, 0] - 7000;
            proporcion = myLmap.GetPROPORCION(height / 5, width / 5);
        }
        else
        {
            proporcion = myLmap.GetPROPORCION(height, width);
        }

        try
        {
            int cont = 0;
            Pen myPen;
            while (cont < myList.GetNumList())
            {
                if (segmentation)
                {
                    if ((myList.GetPlanI(cont).GetZona() == 01) ||
(myList.GetPlanI(cont).GetZona() == 02))
                    {
                        myPen = new Pen(Color.BlueViolet, grosor);
                    }
                    else if ((myList.GetPlanI(cont).GetZona() == 11) ||
(myList.GetPlanI(cont).GetZona() == 12) || (myList.GetPlanI(cont).GetZona() ==
13))
                    {
                        myPen = new Pen(Color.Yellow, grosor);
                    }
                    else if ((myList.GetPlanI(cont).GetZona() == 21) ||
(myList.GetPlanI(cont).GetZona() == 22))
                    {
                        myPen = new Pen(Color.Black, grosor);
                    }
                    else if ((myList.GetPlanI(cont).GetZona() == 31) ||
(myList.GetPlanI(cont).GetZona() == 32))
                    {
                        myPen = new Pen(Color.Red, grosor);
                    }
                    else if (myList.GetPlanI(cont).GetZona() == 4)
                    {
                        myPen = new Pen(Color.Green, grosor);
                    }
                    else
                    {
                        myPen = new Pen(Color.DeepPink, grosor);
                    }
                }
                else myPen = new Pen(Color.Blue, grosor);
```

```
                if ((myList.GetPlanI(cont).GetPosicion()[0] != Math.Pow(10, 8)) &&
(myList.GetPlanI(cont).GetPosicion()[1] != Math.Pow(10, 8)))
                {
                    graphics.DrawRectangle(myPen,
Convert.ToInt32((myList.GetPlanI(cont).GetPosicion()[0] - dimensiones[0, 0]) /
proporcion), Convert.ToInt32((-(myList.GetPlanI(cont).GetPosicion()[1] -
dimensiones[1, 0]) / proporcion) + height), 1, 1);
                }
                cont = cont + 1;
            }
            graphics.Dispose();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    private void PintarAllDGPS(Bitmap Bit, int width, int height)//Pinta todos los
paquetes de myListDGPS
    {
        Graphics graphics;
        graphics = Graphics.FromImage(Bit);
        double[,] dimensiones = myLmap.GetDIMENSIONES();
        double proporcion;

        if (zoom)
        {
            dimensiones[0, 0] = dimensiones[0, 0] - 10000;
            dimensiones[1, 0] = dimensiones[1, 0] - 7000;
            proporcion = myLmap.GetPROPORCION(height / 5, width / 5);
        }
        else
        {
            proporcion = myLmap.GetPROPORCION(height, width);
        }

        try
        {
            int cont = 0;
            Pen myPen;
            Pen myPenMLAT;
            Pen myPenUnion;
            double diferencia;
            while (cont < myListDGPS.GetNumList())
            {
                myPen = new Pen(Color.LimeGreen, grosor);
                myPenMLAT = new Pen(Color.Blue, grosor);
                if (MLATandDGPS)
```

```
            {
                if (myListDGPS.GetPlanI(cont).GetIndiceDGPS() != -1)
                {
                    diferencia =
Math.Sqrt(Math.Pow(myListDGPS.GetPlanI(cont).GetPosicion()[0] -
myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS()).GetPosicion()[0
], 2) + Math.Pow(myListDGPS.GetPlanI(cont).GetPosicion()[1] -
myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS()).GetPosicion()[1
], 2));
                    if (diferencia < 5) //El color de la union cambia segun el error
en posición
                        myPenUnion = new Pen(Color.LawnGreen, grosor);
                    else if (diferencia < 7.5)
                        myPenUnion = new Pen(Color.LightSkyBlue, grosor);
                    else if (diferencia < 12)
                        myPenUnion = new Pen(Color.Orange, grosor);
                    else myPenUnion = new Pen(Color.Red, grosor);
                    graphics.DrawRectangle(myPen,
Convert.ToInt32((myListDGPS.GetPlanI(cont).GetPosicion()[0] - dimensiones[0,
0]) / proporcion), Convert.ToInt32((-
(myListDGPS.GetPlanI(cont).GetPosicion()[1] - dimensiones[1, 0]) / proporcion)
+ height), 1, 1);
                    graphics.DrawRectangle(myPenMLAT,
Convert.ToInt32((myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS(
)).GetPosicion()[0] - dimensiones[0, 0]) / proporcion), Convert.ToInt32((-
(myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS()).GetPosicion()[
1] - dimensiones[1, 0]) / proporcion) + height), 1, 1);
                    graphics.DrawLine(myPenUnion,
Convert.ToInt32((myListDGPS.GetPlanI(cont).GetPosicion()[0] - dimensiones[0,
0]) / proporcion), Convert.ToInt32((-
(myListDGPS.GetPlanI(cont).GetPosicion()[1] - dimensiones[1, 0]) / proporcion)
+ height),
Convert.ToInt32((myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS(
)).GetPosicion()[0] - dimensiones[0, 0]) / proporcion), Convert.ToInt32((-
(myList.GetPlanI(myListDGPS.GetPlanI(cont).GetIndiceDGPS()).GetPosicion()[
1] - dimensiones[1, 0]) / proporcion) + height)); //Linea que une DGPS con
MLAT
                }
            }
            else
            {
                if (segmentation)
                {
                    if ((myListDGPS.GetPlanI(cont).GetZona() == 01) ||
(myListDGPS.GetPlanI(cont).GetZona() == 02))
                    {
                        myPen = new Pen(Color.BlueViolet, grosor);
                    }
```

```
                    else if ((myListDGPS.GetPlanI(cont).GetZona() == 11) ||
(myListDGPS.GetPlanI(cont).GetZona() == 12) ||
(myListDGPS.GetPlanI(cont).GetZona() == 13))
                    {
                        myPen = new Pen(Color.Yellow, grosor);
                    }
                    else if ((myListDGPS.GetPlanI(cont).GetZona() == 21) ||
(myListDGPS.GetPlanI(cont).GetZona() == 22))
                    {
                        myPen = new Pen(Color.Black, grosor);
                    }
                    else if ((myListDGPS.GetPlanI(cont).GetZona() == 31) ||
(myListDGPS.GetPlanI(cont).GetZona() == 32))
                    {
                        myPen = new Pen(Color.Red, grosor);
                    }
                    else if (myListDGPS.GetPlanI(cont).GetZona() == 4)
                    {
                        myPen = new Pen(Color.Green, grosor);
                    }
                    else
                    {
                        myPen = new Pen(Color.DeepPink, grosor);
                    }
                }
                graphics.DrawRectangle(myPen,
Convert.ToInt32((myListDGPS.GetPlanI(cont).GetPosicion()[0] - dimensiones[0,
0]) / proporcion), Convert.ToInt32((-
(myListDGPS.GetPlanI(cont).GetPosicion()[1] - dimensiones[1, 0]) / proporcion)
+ height), 1, 1);
                }
                cont = cont + 1;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    private void PintarAllDescartados(Bitmap Bit, int width, int height, string
filtro)//Pinta todos los paquetes de myListConDescartados
    {
        Graphics graphics;
        graphics = Graphics.FromImage(Bit);
        double[,] dimensiones = dimensionesDescartado;
        double proporcion = proporcionDescartados;

        try
```

```
        {
            int cont = 0;
            Pen myPen = new Pen(Color.Red, grosor);
            while (cont < myListConDescartados.GetNumList())
            {
                if (myListConDescartados.GetPlanI(cont).GetICAOAdress() ==
filtro)
                {
                    if ((myListConDescartados.GetPlanI(cont).GetPosicion()[0] !=
Math.Pow(10, 8)) && (myListConDescartados.GetPlanI(cont).GetPosicion()[1]
!= Math.Pow(10, 8)))
                    {
                        graphics.DrawRectangle(myPen,
Convert.ToInt32((myListConDescartados.GetPlanI(cont).GetPosicion()[0] -
dimensiones[0, 0]) / proporcion), Convert.ToInt32((-
(myListConDescartados.GetPlanI(cont).GetPosicion()[1] - dimensiones[1, 0]) /
proporcion) + height), 1, 1);
                    }
                }
                cont = cont + 1;
            }
            graphics.Dispose();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    public void PintarSecciones(Bitmap Bit, int width, int height) //Leyendo el
.txt de secciones lo pinta sobre el mapa
    {
        Graphics graphics;
        graphics = Graphics.FromImage(Bit);
        double[,] dimensiones = myLmap.GetDIMENSIONES();
        double proporcion;
        if (zoom)
        {
            dimensiones[0, 0] = dimensiones[0, 0] - 10000;
            dimensiones[1, 0] = dimensiones[1, 0] - 7000;
            proporcion = myLmap.GetPROPORCION(height / 5, width / 5);
        }
        else
        {
            proporcion = myLmap.GetPROPORCION(height, width);
        }

        string nombreFichero = "Secciones";
        StreamReader leer = new StreamReader(nombreFichero + ".txt");
```

```
            string linea = leer.ReadLine();
            int numZona = 1;
            while (linea != null)
            {
                List<Point> points = new List<Point>();
                while (linea != "/")
                {
                    string[] coordenadas = linea.Split(' ');
                    int X = Convert.ToInt32((Convert.ToDouble(coordenadas[0]) -
dimensiones[0, 0]) / proporcion);
                    int Y = Convert.ToInt32((-(Convert.ToDouble(coordenadas[1]) -
dimensiones[1, 0]) / proporcion) + height);
                    points.Add(new Point(X, Y));
                    linea = leer.ReadLine();
                }
                Point[] Point = new Point[points.Count()];
                for (int i = 0; i < points.Count(); i++)
                    Point[i] = points[i];

                int opacidad = 50;//De 0 a 255
                Color myColor = new Color();
                if ((numZona >= 1) && (numZona <= 5))//RWY
                    myColor = Color.FromArgb(opacidad, Color.Yellow);
                else if ((numZona >= 6) && (numZona <= 16))//2 stand
                    myColor = Color.FromArgb(opacidad, Color.Black);
                else if ((numZona >= 17) && (numZona <= 25))//3 apron
                    myColor = Color.FromArgb(opacidad, Color.Red);
                else if ((numZona >= 26) && (numZona <= 36))//4 taxi
                    myColor = Color.FromArgb(opacidad, Color.Green);
                else if ((numZona >= 37) && (numZona <= 48))//01,02 areas tipo 4,5
                    myColor = Color.FromArgb(opacidad, Color.BlueViolet);

                graphics.FillPolygon(new SolidBrush(myColor), Point);
                points.Clear();
                linea = leer.ReadLine();
                numZona++;
            }
            graphics.Dispose();
            leer.Close();
        }

        private Bitmap SaveMap()//Prepara el Bitmap para ser exportado
        {
            int[] calidad;
            if (zoom)
            {
                calidad = new int[2] { 6000, 4400 }; //en PintarAll el grosor se pone en
2 si calidad[0]=6000
                grosor = 2;
            }
```

```
            else
            {
                calidad = new int[2] { 2500, 2200 };
                grosor = 3;
            }
            Bitmap saveBitmap = new Bitmap(calidad[0], calidad[1],
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(saveBitmap, calidad[0], calidad[1]);
            if (DGPS)
            {
                if (!MLATandDGPS)
                {
                    PintarAllDGPS(saveBitmap, calidad[0], calidad[1]);
                }
                else PintarAllDGPS(saveBitmap, calidad[0], calidad[1]);
            }
            else if (MLATandDGPS)
            {
                PintarAllDGPS(saveBitmap, calidad[0], calidad[1]); //Pinta los
paquetes DGPS sobre el mapa
            }
            else PintarAll(saveBitmap, calidad[0], calidad[1]);

            if (segmentation)
            {
                PintarSecciones(saveBitmap, calidad[0], calidad[1]);
            }
            return (saveBitmap);
        }

        private void ButtonSaveMap_Click(object sender, EventArgs e)//Boton
exportar el mapa LEBL a .png
        {
            Bitmap saveBitmap = SaveMap();

            string saveName = nombreFichero;//Añadimos caracteristicas actuales
al nombre del fichero a guardar
            if (MLATandDGPS)
                saveName = saveName + "_MLATandDGPS";
            else if (!DGPS)
                saveName = saveName + "_MLAT";
            else if (DGPS)
                saveName = saveName + "_DGPS";

            if (segmentation)
                saveName = saveName + "_Segmented";
            if (zoom)
                saveName = saveName + "_ZoomOut";
            if (HideBack)
                saveName = saveName + "_WithOutBackground";
```

```
            saveFileDialog1.FileName = saveName + "_Map.png";
            saveFileDialog1.Filter = "File documents (.png)|*.png";
            saveFileDialog1.ShowDialog();
            try
            {
                if (this.saveFileDialog1.FileName.Equals("") == false)
                {
                    if (this.saveFileDialog1.FileName.Contains(@"\"))//Controla si se
ha escogido un directorio o se ha cancelado/cerrado el save dialog
                    {
                        Cursor.Current = Cursors.WaitCursor;
                        saveBitmap.Save(saveFileDialog1.FileName);
                        saveAllCorrectoPorSeparado[0] = true;
                        if (saveAllCorrectoPorSeparado[1])//Si el del excel tambien es
true, el general tambien pasa a ser true
                            saveAllCorrecto = true;
                    }
                }
                else CrearFormInformativa("Please write a", "name first.", 187, 30, 27,
41);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
                return;
            }
        }

        private void panelMap_MouseClick(object sender, MouseEventArgs e)
//Devuelve la posicion respecto MLAT al clicar en pantalla
        {
            int xp = e.X;
            int yp = e.Y;
            double[,] dimensiones = myLmap.GetDIMENSIONES();
            double proporcion;
            if (zoom)
            {
                dimensiones[0, 0] = dimensiones[0, 0] - 10000;
                dimensiones[1, 0] = dimensiones[1, 0] - 7000;
                proporcion = myLmap.GetPROPORCION(this.panelMap.Height / 5,
this.panelMap.Width / 5);
            }
            else
            {
                proporcion = myLmap.GetPROPORCION(this.panelMap.Height,
this.panelMap.Width);
            }

            double Xr = Math.Round(xp * proporcion + dimensiones[0, 0], 2);
```

```
            double Yr = Math.Round(dimensiones[1, 0] + (this.panelMap.Height -
yp) * proporcion, 2);
            labelCursor.Text = "Cursor: " + Xr + "m, " + Yr + "m (X,Y)";
        }

        private void panelMap_MouseMove(object sender, MouseEventArgs
e)//Estblece los valores X, Y del raton sobre el mapa
        {
            int xp = e.X;
            int yp = e.Y;
            double[,] dimensiones = myLmap.GetDIMENSIONES();
            doubl e proporcion;
            if (zoom)
            {
                dimensiones[0, 0] = dimensiones[0, 0] - 10000;
                dimensiones[1, 0] = dimensiones[1, 0] - 7000;
                proporcion = myLmap.GetPROPORCION(this.panelMap.Height / 5,
this.panelMap.Width / 5);
            }
            else
            {
                proporcion = myLmap.GetPROPORCION(this.panelMap.Height,
this.panelMap.Width);
            }

            double Xr = Math.Round(xp * proporcion + dimensiones[0, 0], 2);
            double Yr = Math.Round(dimensiones[1, 0] + (this.panelMap.Height -
yp) * proporcion, 2);
            labelCursor.Text = "Cursor: " + Xr + "m, " + Yr + "m (X,Y)";
        }

        private void panelMap_MouseLeave(object sender, EventArgs e)//Pone el
cursor a 0 cuando se abandona el panel
        {
            labelCursor.Text = "Cursor: --.--m, --.--m (X,Y)";
        }

        private void buttonViewSegmentation_Click(object sender, EventArgs
e)//Mostrar las zonas del mapa
        {
            grosor = 1;
            if (!segmentation)
            {
                segmentation = true;
                buttonViewSegmentation.Text = "Hide Segmentation";
                myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
```

```
            PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            if (DGPS)
            {
                if (!MLATandDGPS)
                {
                    PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                }
                else PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            }
            else if (MLATandDGPS)
            {
                PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
            }
            else PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);

            panelMap.Invalidate();
        }
        else
        {
            buttonViewSegmentation.Text = "View Segmentation";
            segmentation = false;
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            if (DGPS)
            {
                if (!MLATandDGPS)
                {
                    PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                }
                else PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            }
            else if (MLATandDGPS)
            {
                PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
            }
            else PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);

            panelMap.Invalidate();
        }
    }
```

```
        private void ButtonZoom_Click(object sender, EventArgs e) //Hace zoom in
y zoom out en el mapa
        {
            grosor = 1;
            if (!zoom)
            {
                zoom = true;
                ButtonZoom.Text = "Zoom In";
                myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                if (DGPS)
                {
                    if (!MLATandDGPS)
                    {
                        PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                    }
                    else PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                }
                else if (MLATandDGPS)
                {
                    PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
                }
                else PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);

                if  (segmentation)
                {
                    PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
                }
                panelMap.Invalidate();
            }
            else
            {
                zoom = false;
                ButtonZoom.Text = "Zoom Out";
                myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                if (DGPS)
                {
                    if (!MLATandDGPS)
                    {
                        PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
```

```
                }
                else PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            }
            else if (MLATandDGPS)
            {
                PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
            }
            else PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);

            if (segmentation)
            {
                PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            }
            panelMap.Invalidate();
        }
    }

    private void ButtonDGPS_Click(object sender, EventArgs e) //Muestra solo
el mensaje DGPS
    {
        MLATandDGPS = false;
        grosor = 1;

        if (!DGPS)
        {
            DGPS = true;
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            if (segmentation)
            {
                PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            }
            panelMap.Invalidate();
            ButtonDGPS.Text = "View only MLAT";
        }
        else
        {
            DGPS = false;
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
```

```
            PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);
//Pinta los paquetes sobre el mapa
            if (segmentation)
            {
                PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            }
            panelMap.Invalidate();
            ButtonDGPS.Text = "View only D-GPS";
        }
    }

    private void ButtonMLAT_DGPS_Click(object sender, EventArgs e)
//Muestra el mensaje DGPS y el trafico MLAT correspondiente.
    {
        grosor = 1;
        if (!MLATandDGPS)
        {
            MLATandDGPS = true;
            DGPS = true;
            ButtonDGPS.Text = "View only MLAT";
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
            if (segmentation)
            {
                PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            }
            panelMap.Invalidate();
        }
    }

    private void buttonHideBackground_Click_1(object sender, EventArgs
e)//Esconde el mapa de LEBL
    {
        grosor = 1;
        if (!HideBack)
        {
            HideBack = true;
            buttonHideBackground.Text = "Show Background";
        }
        else
        {
            HideBack = false;
            buttonHideBackground.Text = "Hide Background";
        }
```

```
            myBitmap = new Bitmap(this.panelMap.Width, this.panelMap.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            if (segmentation)
            {
                PintarSecciones(myBitmap, this.panelMap.Width,
this.panelMap.Height);//Pinta encima del mapa las diferentes secciones
            }
            if (DGPS)
            {
                if (!MLATandDGPS)
                {
                    PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
                }
                else PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height);
            }
            else if (MLATandDGPS)
            {
                PintarAllDGPS(myBitmap, this.panelMap.Width,
this.panelMap.Height); //Pinta los paquetes DGPS sobre el mapa
            }
            else PintarAll(myBitmap, this.panelMap.Width, this.panelMap.Height);

            panelMap.Invalidate();
        }

        private void CrearTablaSummary(DataGridView grid, Avaluador avaluador)
//Update Rate
        {
            grid.RowCount = avaluador.GetUpdateRate().GetLength(0);
            grid.ColumnCount = avaluador.GetUpdateRate().GetLength(1);
            grid.ColumnHeadersVisible = true;
            grid.RowHeadersVisible = true;
            grid.ClearSelection();
            for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
            {
                grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
            }
            Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
            grid.Columns[0].Name = "Updates";
            grid.Columns[1].Name = "Expected";
            grid.Columns[2].Name = "UR (%) (updates/s)";
            grid.Columns[3].Name = "Minimum UR (%)";
            grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
```

```
            grid.Rows[1].HeaderCell.Value = "RWY 25L";
            grid.Rows[2].HeaderCell.Value = "RWY 02";
            grid.Rows[3].HeaderCell.Value = "RWY 25R";
            grid.Rows[4].HeaderCell.Value = "Taxi";
            grid.Rows[5].HeaderCell.Value = "Apron";
            grid.Rows[6].HeaderCell.Value = "Apron T1";
            grid.Rows[7].HeaderCell.Value = "Apron T2";
            grid.Rows[8].HeaderCell.Value = "Stands";
            grid.Rows[9].HeaderCell.Value = "Stands T1";
            grid.Rows[10].HeaderCell.Value = "Stands T2";
            grid.Rows[11].HeaderCell.Value = "Airborne";
            grid.Rows[12].HeaderCell.Value = "Type 4";
            grid.Rows[13].HeaderCell.Value = "Type 5";
            grid.Rows[14].HeaderCell.Value = "Rest";
            grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[7].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[10].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[12].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[13].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[14].HeaderCell.Style.Padding = paddingZonasPrincipales;
            for (int i = 0; i < avaluador.GetUpdateRate().GetLength(0); i++)
            {
                for (int j = 0; j < avaluador.GetUpdateRate().GetLength(1); j++)
                {
                    if ((j == 2) || (j == 3))//Redondea las columnas UR y minimum UR
                    {
                        grid[j, i].Value = Math.Round(avaluador.GetUpdateRate()[i, j], 3);
                    }
                    else grid[j, i].Value = Math.Floor(avaluador.GetUpdateRate()[i,
j]);//Redondea a la baja los expected updates

                    if (j == 2)//Si UR = 0, no hay UR
                    {
                        if (avaluador.GetUpdateRate()[i, j] == 0)
                        {
                            grid[j, i].Value = null;
                        }
                    }

                    if ((i == 0) || (i == 5) || (i == 8) || (i == 11))//Controla si el fondo es
verde o rojo segun cumplan o no con el minimo
                    {
                        if (j == 2)
                        {
                            if (avaluador.GetUpdateRate()[i, j] != 0)
```

```
                    {
                        if (avaluador.GetUpdateRate()[i, j] >=
avaluador.GetUpdateRate()[i, j + 1])
                        {
                            grid[j, i].Style.BackColor = Color.Green;
                        }
                        else grid[j, i].Style.BackColor = Color.Red;
                    }
                }
            }

            Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
            if ((i == 0) || (i == 5) || (i == 8) || (i == 11))
            {
                grid[j, i].Style.Font = fuentePrincipal;
            }
            else
            {
                if (j == 3)//Quitamos el 0 de la columna minimum value para las
filas no principles;
                {
                    grid[j, i].Value = null;
                }
            }
        }
    }
}

    private void CrearTablaPositionAccuracy(DataGridView grid, Avaluador
avaluador)  //Position Accuracy
    {
        grid.RowCount = avaluador.GetPositionAccuracy().GetLength(0);
        grid.ColumnCount = avaluador.GetPositionAccuracy().GetLength(1);
        grid.ColumnHeadersVisible = true;
        grid.RowHeadersVisible = true;
        grid.ClearSelection();
        for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
        {
            grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
        }
        Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
        grid.Columns[0].Name = "P95";
        grid.Columns[1].Name = "P99";
        grid.Columns[2].Name = "Max Detected";
        grid.Columns[3].Name = "max P95";
        grid.Columns[4].Name = "max P99";
        grid.Columns[5].Name = "Mean";
```

```
            grid.Columns[6].Name = "STD";
            grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
            grid.Rows[1].HeaderCell.Value = "RWY 25L";
            grid.Rows[2].HeaderCell.Value = "RWY 02";
            grid.Rows[3].HeaderCell.Value = "RWY 25R";
            grid.Rows[4].HeaderCell.Value = "Taxi";
            grid.Rows[5].HeaderCell.Value = "Apron";
            grid.Rows[6].HeaderCell.Value = "Apron T1";
            grid.Rows[7].HeaderCell.Value = "Apron T2";
            grid.Rows[8].HeaderCell.Value = "Stands";
            grid.Rows[9].HeaderCell.Value = "Stands T1";
            grid.Rows[10].HeaderCell.Value = "Stands T2";
            grid.Rows[11].HeaderCell.Value = "Airborne (No Data)";
            grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[7].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[10].HeaderCell.Style.Padding = paddingZonasPrincipales;
            for (int i = 0; i < avaluador.GetPositionAccuracy().GetLength(0); i++)
            {
                for (int j = 0; j < avaluador.GetPositionAccuracy().GetLength(1); j++)
                {
                    if ((j == 0) || (j == 1) || (j == 2) || (j == 5) || (j == 6))//Redondea las
columnas P95 y P99, max value, mean y STD
                    {
                        grid[j, i].Value = Math.Round(avaluador.GetPositionAccuracy()[i,
j], 2);
                    }
                    else grid[j, i].Value = avaluador.GetPositionAccuracy()[i, j];

                    if ((i == 0) || (i == 5))//Controla si el fondo es verde o rojo segun
cumplan o no con el minimo
                    {
                        if ((j == 0)||(j==1))
                        {
                            if (avaluador.GetPositionAccuracy()[i, j] !=-1)
                            {
                                if (avaluador.GetPositionAccuracy()[i, j] <=
avaluador.GetPositionAccuracy()[i, j + 3])
                                {
                                    grid[j, i].Style.BackColor = Color.Green;
                                }
                                else grid[j, i].Style.BackColor = Color.Red;
                            }
                        }
                    }
                    else
```

```
                {
                    if ((j == 3) || (j == 4))//Quitamos el 0 de la columna minimum
value para las filas no principles
                    {
                        grid[j, i].Value = null;
                    }
                }

                if (i == 8)//Controla si el fondo es verde o rojo segun cumplan o no
con el minimo
                {
                    if (j == 2)
                    {
                        if (avaluador.GetPositionAccuracy()[i, j] != -1)
                        {
                            if (avaluador.GetPositionAccuracy()[i, j] <= 20)//max 20m en
Stand PosAcc
                            {
                                grid[j, i].Style.BackColor = Color.Green;
                            }
                            else grid[j, i].Style.BackColor = Color.Red;
                        }
                    }
                }

                Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
                if ((i == 0) || (i == 5) || (i == 8) || (i == 11))
                {
                    grid[j, i].Style.Font = fuentePrincipal;
                }

                if(avaluador.GetPositionAccuracy()[i, j] == -1)//Quitamos el -1 de
las zonas que no tenemos informacion
                {
                    grid[j, i].Value = null;
                }
            }
        }
    }

    private void CrearTablaProbOfMLATDetection(DataGridView grid,
Avaluador avaluador)
    {
        grid.RowCount = avaluador.GetProbOfMLATDetection().GetLength(0);
        grid.ColumnCount =
avaluador.GetProbOfMLATDetection().GetLength(1);
        grid.ColumnHeadersVisible = true;
        grid.RowHeadersVisible = true;
        grid.ClearSelection();
```

```
        for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
        {
            grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
        }
        Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
        grid.Columns[0].Name = "Detected";
        grid.Columns[1].Name = "Expected";
        grid.Columns[2].Name = "PD (%) (Prob. of Detection)";
        grid.Columns[3].Name = "Minimum PD (%)";
        grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
        grid.Rows[1].HeaderCell.Value = "RWY 25L";
        grid.Rows[2].HeaderCell.Value = "RWY 02";
        grid.Rows[3].HeaderCell.Value = "RWY 25R";
        grid.Rows[4].HeaderCell.Value = "Taxi";
        grid.Rows[5].HeaderCell.Value = "Apron T1";
        grid.Rows[6].HeaderCell.Value = "Apron T2";
        grid.Rows[7].HeaderCell.Value = "Stands";
        grid.Rows[8].HeaderCell.Value = "Stands T1";
        grid.Rows[9].HeaderCell.Value = "Stands T2";
        grid.Rows[10].HeaderCell.Value = "Airborne (Not Required)";
        grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[5].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[8].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
        for (int i = 0; i < avaluador.GetProbOfMLATDetection().GetLength(0);
i++)
        {
            for (int j = 0; j < avaluador.GetProbOfMLATDetection().GetLength(1);
j++)
            {
                if ((j == 2) || (j == 3))//Redondea las columnas UR y minimum UR
                {
                    grid[j, i].Value =
Math.Round(avaluador.GetProbOfMLATDetection()[i, j], 3);
                }
                else grid[j, i].Value = avaluador.GetProbOfMLATDetection()[i, j];

                if (j == 2)//Si PD = 0, no hay PD
                {
                    if (avaluador.GetProbOfMLATDetection()[i, j] == 0)
                    {
                        grid[j, i].Value = null;
                    }
                }
```

```
                Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
                if ((i == 0) || (i == 7))//Controla si el fondo es verde o rojo segun
cumplan o no con el minimo
                {
                    grid[j, i].Style.Font = fuentePrincipal;
                    if (j == 2)
                    {
                        if (avaluador.GetProbOfMLATDetection()[i, j] != 0)
                        {
                            if (avaluador.GetProbOfMLATDetection()[i, j] >=
avaluador.GetProbOfMLATDetection()[i, j + 1])
                            {
                                grid[j, i].Style.BackColor = Color.Green;
                            }
                            else grid[j, i].Style.BackColor = Color.Red;
                        }
                    }
                }
                else if (j == 3)//Quitamos el 0 de la columna minimum value para
las filas no principles;
                {
                    grid[j, i].Value = null;
                }

                if (i == 10) //Quitamos el 0 de Apron y Airborne
                {
                    grid[j, i].Value = null;
                }
            }
        }
    }

    private void CrearTablaProbOfIdentification(DataGridView grid, Avaluador
avaluador)
    {
        grid.RowCount = avaluador.GetProbOfIdentification().GetLength(0);
        grid.ColumnCount = avaluador.GetProbOfIdentification().GetLength(1);
        grid.ColumnHeadersVisible = true;
        grid.RowHeadersVisible = true;
        grid.ClearSelection();
        for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
        {
            grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
        }
        Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
        grid.Columns[0].Name = "Correct";
```

```
            grid.Columns[1].Name = "Inorrect";
            grid.Columns[2].Name = "PID (%) (correct/total)";
            grid.Columns[3].Name = "Minimum PID (%)";
            grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
            grid.Rows[1].HeaderCell.Value = "RWY 25L";
            grid.Rows[2].HeaderCell.Value = "RWY 02";
            grid.Rows[3].HeaderCell.Value = "RWY 25R";
            grid.Rows[4].HeaderCell.Value = "Taxi";
            grid.Rows[5].HeaderCell.Value = "Apron";
            grid.Rows[6].HeaderCell.Value = "Apron T1";
            grid.Rows[7].HeaderCell.Value = "Apron T2";
            grid.Rows[8].HeaderCell.Value = "Stands";
            grid.Rows[9].HeaderCell.Value = "Stands T1";
            grid.Rows[10].HeaderCell.Value = "Stands T2";
            grid.Rows[11].HeaderCell.Value = "Airborne";
            grid.Rows[12].HeaderCell.Value = "Type 4";
            grid.Rows[13].HeaderCell.Value = "Type 5";
            grid.Rows[14].HeaderCell.Value = "Rest";
            grid.Rows[15].HeaderCell.Value = "Total";
            grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[7].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[10].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[12].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[13].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[14].HeaderCell.Style.Padding = paddingZonasPrincipales;
            for (int i = 0; i < avaluador.GetProbOfIdentification().GetLength(0); i++)
            {
                for (int j = 0; j < avaluador.GetProbOfIdentification().GetLength(1);
j++)
                {
                    grid[j, i].Value = avaluador.GetProbOfIdentification()[i, j];

                    if (i == 15)//Controla si el fondo es verde o rojo segun cumplan o no
con el minimo
                    {
                        if (j == 2)
                        {
                            if (avaluador.GetProbOfIdentification()[i, j - 2] != 0)
                            {
                                if (avaluador.GetProbOfIdentification()[i, j] >=
avaluador.GetProbOfIdentification()[i, j + 1])
                                {
                                    grid[j, i].Style.BackColor = Color.Green;
                                }
                                else grid[j, i].Style.BackColor = Color.Red;
```

```
                    }
                }
            }

            if (j == 2)//Redondeamos columna PID
            {
                grid[j, i].Value =
Math.Round(avaluador.GetProbOfIdentification()[i, j], 3);
            }

            Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
            if ((i == 0) || (i == 5) || (i == 8) || (i == 11) || (i == 15))
            {
                grid[j, i].Style.Font = fuentePrincipal;
            }

            if (i != 15)
            {
                if (j == 3)//Quitamos el 0 de la columna minimum value para
todas las filas menos total;
                {
                    grid[j, i].Value = null;
                }
            }
        }
    }
    for (int i = 0; i < avaluador.GetProbOfIdentification().GetLength(0);
i++)//Si correcto = 0, no hay PID
    {
        if (avaluador.GetProbOfIdentification()[i, 0] == 0)
        {
            grid[2, i].Value = null;
        }
    }
}

private void CrearTablaProbOfFalseDetection(DataGridView grid,
Avaluador avaluador)
{
    grid.RowCount = avaluador.GetProbOfFalseDetection().GetLength(0);
    grid.ColumnCount =
avaluador.GetProbOfFalseDetection().GetLength(1);
    grid.ColumnHeadersVisible = true;
    grid.RowHeadersVisible = true;
    grid.ClearSelection();
    for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
    {
```

```
            grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
        }
        Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
        grid.Columns[0].Name = "Reports";
        grid.Columns[1].Name = "False Reports";
        grid.Columns[2].Name = "PFD (%) (False/Reports)";
        grid.Columns[3].Name = "Minimum PFD (%)";
        grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
        grid.Rows[1].HeaderCell.Value = "RWY 25L";
        grid.Rows[2].HeaderCell.Value = "RWY 02";
        grid.Rows[3].HeaderCell.Value = "RWY 25R";
        grid.Rows[4].HeaderCell.Value = "Taxi";
        grid.Rows[5].HeaderCell.Value = "Apron";
        grid.Rows[6].HeaderCell.Value = "Apron T1";
        grid.Rows[7].HeaderCell.Value = "Apron T2";
        grid.Rows[8].HeaderCell.Value = "Stands";
        grid.Rows[9].HeaderCell.Value = "Stands T1";
        grid.Rows[10].HeaderCell.Value = "Stands T2";
        grid.Rows[11].HeaderCell.Value = "Airborne";
        grid.Rows[12].HeaderCell.Value = "Type 4";
        grid.Rows[13].HeaderCell.Value = "Type 5";
        grid.Rows[14].HeaderCell.Value = "Total";
        grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[7].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[10].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[12].HeaderCell.Style.Padding = paddingZonasPrincipales;
        grid.Rows[13].HeaderCell.Style.Padding = paddingZonasPrincipales;
        for (int i = 0; i < avaluador.GetProbOfFalseDetection().GetLength(0);
i++)
        {
            for (int j = 0; j < avaluador.GetProbOfFalseDetection().GetLength(1);
j++)
            {
                grid[j, i].Value = avaluador.GetProbOfFalseDetection()[i, j];

                if (i == 14)//Controla si el fondo es verde o rojo segun cumplan o no
con el minimo
                {
                    if (j == 2)
                    {
                        if (avaluador.GetProbOfFalseDetection()[i, j - 2] != 0)
                        {
                            if (avaluador.GetProbOfFalseDetection()[i, j] <=
avaluador.GetProbOfFalseDetection()[i, j + 1])
```

```
                        {
                            grid[j, i].Style.BackColor = Color.Green;
                        }
                        else grid[j, i].Style.BackColor = Color.Red;
                    }
                }
            }

            if (j == 2)//Redondeamos columna PFD, añadimos simbolo %
            {
                grid[j, i].Value =
Math.Round(avaluador.GetProbOfFalseDetection()[i, j], 3);
            }

            Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
            if ((i == 0) || (i == 5) || (i == 8) || (i == 11) || (i == 14))
            {
                grid[j, i].Style.Font = fuentePrincipal;
            }

            if (j == 3)//Quitamos el 0 de la columna minimum value para todas
las filas menos total;
            {
                if (i != 14)
                {
                    grid[j, i].Value = null;
                }
                else grid[j, i].Value = grid[j, i].Value;
            }
        }
    }
    for (int i = 0; i < avaluador.GetProbOfFalseDetection().GetLength(0);
i++)//Si reports = 0, no hay ProbFalseDet
    {
        if (avaluador.GetProbOfFalseDetection()[i, 0] == 0)
        {
            grid[2, i].Value = null;
        }
    }
}

private void CrearTablaProbOfFalseIdentification(DataGridView grid,
Avaluador avaluador)
{
    grid.RowCount =
avaluador.GetProbOfFalseIdentification().GetLength(0);
    grid.ColumnCount =
avaluador.GetProbOfFalseIdentification().GetLength(1);
    grid.ColumnHeadersVisible = true;
```

```
            grid.RowHeadersVisible = true;
            grid.ClearSelection();
            for (int i = 0; i < grid.ColumnCount; i++)//Desactivamos la funcion de
SortMode
            {
                grid.Columns[i].SortMode =
DataGridViewColumnSortMode.Programmatic;
            }
            Padding paddingZonasPrincipales = new Padding(20, 0, 0, 0);
            grid.Columns[0].Name = "Total";
            grid.Columns[1].Name = "False";
            grid.Columns[2].Name = "Prob. False ID (%) (false/total)";
            grid.Columns[3].Name = "Minimum Prob. False ID (%)";
            grid.Rows[0].HeaderCell.Value = "Maneouvering Area";
            grid.Rows[1].HeaderCell.Value = "RWY 25L";
            grid.Rows[2].HeaderCell.Value = "RWY 02";
            grid.Rows[3].HeaderCell.Value = "RWY 25R";
            grid.Rows[4].HeaderCell.Value = "Taxi";
            grid.Rows[5].HeaderCell.Value = "Apron";
            grid.Rows[6].HeaderCell.Value = "Apron T1";
            grid.Rows[7].HeaderCell.Value = "Apron T2";
            grid.Rows[8].HeaderCell.Value = "Stands";
            grid.Rows[9].HeaderCell.Value = "Stands T1";
            grid.Rows[10].HeaderCell.Value = "Stands T2";
            grid.Rows[11].HeaderCell.Value = "Airborne";
            grid.Rows[12].HeaderCell.Value = "Type 4";
            grid.Rows[13].HeaderCell.Value = "Type 5";
            grid.Rows[14].HeaderCell.Value = "Rest";
            grid.Rows[15].HeaderCell.Value = "Total";
            grid.Rows[1].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[2].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[3].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[4].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[6].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[7].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[9].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[10].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[12].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[13].HeaderCell.Style.Padding = paddingZonasPrincipales;
            grid.Rows[14].HeaderCell.Style.Padding = paddingZonasPrincipales;
            for (int i = 0; i < avaluador.GetProbOfFalseIdentification().GetLength(0);
i++)
            {
                for (int j = 0; j <
avaluador.GetProbOfFalseIdentification().GetLength(1); j++)
                {
                    grid[j, i].Value = avaluador.GetProbOfFalseIdentification()[i, j];

                    if (i == 15)//Controla si el fondo es verde o rojo segun cumplan o no
con el minimo
```

```
                {
                    if (j == 2)
                    {
                        if (avaluador.GetProbOfFalseIdentification()[i, j - 2] != 0)
                        {
                            if (avaluador.GetProbOfFalseIdentification()[i, j] <=
avaluador.GetProbOfFalseIdentification()[i, j + 1])
                            {
                                grid[j, i].Style.BackColor = Color.Green;
                            }
                            else grid[j, i].Style.BackColor = Color.Red;
                        }
                    }
                }

                if (j == 2)//Redondeamos columna PFI
                {
                    grid[j, i].Value =
Math.Round(avaluador.GetProbOfFalseIdentification()[i, j], 3);
                }

                Font fuentePrincipal = new Font("Century Gothic", 12,
FontStyle.Bold);//Ponemos en negrita las filas principales
                if ((i == 0) || (i == 5) || (i == 8) || (i == 11) || (i == 15))
                {
                    grid[j, i].Style.Font = fuentePrincipal;
                }

                if (i != 15)
                {
                    if (j == 3)//Quitamos el 0 de la columna minimum value para
todas las filas menos total;
                    {
                        grid[j, i].Value = null;
                    }
                }
            }
        }
        for (int i = 0; i < avaluador.GetProbOfFalseIdentification().GetLength(0);
i++)//Si reports = 0, no hay ProbFalseDet
        {
            if (avaluador.GetProbOfFalseIdentification()[i, 0] == 0)
            {
                grid[2, i].Value = null;
            }
        }
    }

    private void CrearTablaParameters()
    {
```

```
            if (!DGPSresults)
            {
                ParametersGrid.RowCount = 5;
                ParametersGrid.ColumnCount = 1;
                ParametersGrid.ClearSelection();
                ParametersGrid[0, 0].Selected = true;
                ParametersGrid[0, 0].Value = "Update Rate";
                ParametersGrid[0, 1].Value = "Probability of MLAT detection";
                ParametersGrid[0, 2].Value = "Probability of Identification";
                ParametersGrid[0, 3].Value = "Probability of False Detection";
                ParametersGrid[0, 4].Value = "Probability of False Identification";
            }
            else
            {
                ParametersGrid.RowCount = 6;
                ParametersGrid.ColumnCount = 1;
                ParametersGrid.ClearSelection();
                ParametersGrid[0, 0].Selected = true;
                ParametersGrid[0, 0].Value = "Update Rate";
                ParametersGrid[0, 1].Value = "Position Accuracy";
                ParametersGrid[0, 2].Value = "Probability of MLAT detection";
                ParametersGrid[0, 3].Value = "Probability of Identification";
                ParametersGrid[0, 4].Value = "Probability of False Detection";
                ParametersGrid[0, 5].Value = "Probability of False Identification";
            }

        }

        private void ParametersGrid_CellClick(object sender,
DataGridViewCellEventArgs e)//Cambia la summary grid cuando se clica otra
en parameters grid
        {
            int i = e.RowIndex;
            if (i != -1)
            {
                ParametersGrid.ClearSelection();
                ParametersGrid[0, i].Selected = true;
                LabelTituloSummary.Text = Convert.ToString(ParametersGrid[0,
i].Value) + ":";
                SummaryGrid.Rows.Clear();
                SummaryGrid.Columns.Clear();
                if (DGPSresults)
                {
                    if (i == 0)
                        CrearTablaSummary(this.SummaryGrid, myAvaluadorDGPS);
                    else if (i == 1)
                        CrearTablaPositionAccuracy(this.SummaryGrid,
myAvaluadorDGPS);
                    else if (i == 2)
```

```
                    CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 3)
                    CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 4)
                    CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 5)
                    CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluadorDGPS);
            }
            else
            {
                if (i == 0)
                    CrearTablaSummary(this.SummaryGrid, myAvaluador);
                else if (i == 1)
                    CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluador);
                else if (i == 2)
                    CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluador);
                else if (i == 3)
                    CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluador);
                else if (i == 4)
                    CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluador);
            }
        }
    }

    private void panelContenedorResultados_MouseClick(object sender,
MouseEventArgs e)//Quita la seleccion de summary grid si se clica fuera de ella
    {
        //Hay enlace de panelContenedor click a este evento
        SummaryGrid.ClearSelection();
    }

    private void SummaryGrid_Click(object sender, EventArgs e)//Quita la
seleccion de summary grid si se clica en la tabla
    {
        //Si se clica en una celda, primero se activa este evento y despues el
evento de clicar la celda
        SummaryGrid.ClearSelection();
    }

    private void SummaryGrid_CellClick(object sender,
DataGridViewCellEventArgs e)//Si se clica en una celda de summary grid, la
selcciona
```

```
        {
            SummaryGrid[e.ColumnIndex, e.RowIndex].Selected = true;
        }

        private void DGPSresultsButton_Click(object sender, EventArgs
e)//Controla si se muestran los resultados MLAT o D-GPS
        {
            if(!DGPSresults)
            {
                DGPSresults = true;
                DGPSresultsButton.Text = "View MLAT Results";
                SummaryGrid.Rows.Clear();
                SummaryGrid.Columns.Clear();
                CrearTablaSummary(this.SummaryGrid, myAvaluadorDGPS);
                ParametersGrid.Rows.Clear();
                ParametersGrid.Columns.Clear();
                CrearTablaParameters();
                LabelTituloSummary.Text = "Update Rate:";
            }
            else
            {
                DGPSresults = false;
                DGPSresultsButton.Text = "View D-GPS Results";
                SummaryGrid.Rows.Clear();
                SummaryGrid.Columns.Clear();
                CrearTablaSummary(this.SummaryGrid, myAvaluador);
                ParametersGrid.Rows.Clear();
                ParametersGrid.Columns.Clear();
                CrearTablaParameters();
                LabelTituloSummary.Text = "Update Rate:";
            }
        }

        private SLDocument SaveScatter()//prepara el excel para ser exportado;
Exporta solo una tabla (un parametro)
        {
            SLDocument myExcel = new SLDocument();
            myExcel.SetCellValue("A1", "File: " + nombreFichero);//Nombre fichero

            string avionesDetectados = "";
            if (myList.AircraftDetected() != 0)
                avionesDetectados = " from " + myList.AircraftDetected() + " different
transponders ";//Esta frase solo aparece si hay MLAT
            myExcel.SetCellValue("A2", myList.GetNumList() + " packages
avaluated" + avionesDetectados + "between " +
myList.GetPlanI(0).ConvertUTC("SinDecimas") +
                        " hours and " + myList.GetPlanI(myList.GetNumList() -
1).ConvertUTC("SinDecimas") + " hours");//Información del fichero

            myExcel.SetCellValue("A3", "Scatter:");//Titulo de la grid
```

```
        DataTable data = new DataTable();
        data.Columns.Add("X error", typeof(double));//La informacion que
contiene esta columna es de tipo double
        data.Columns.Add("Y error", typeof(double));
        List<double[]> diferencias = myAvaluadorDGPS.GetDiferencias();
        int i;
        int columnaActual = 1;

        myExcel.SetCellValue(4, columnaActual, "RWY 25L");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 11)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);//Inserta la tabla
en la posicion i,j=5,1 (la primera celda en excel es la 1,1 ó A1
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "RWY 02");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 12)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "RWY 25R");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 13)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "Taxi");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 4)
```

```
    {
        data.Rows.Add(diferencias[i][0], diferencias[i][1]);
    }
}
myExcel.ImportDataTable(5, columnaActual, data, true);
data.Rows.Clear();
columnaActual = columnaActual + 2;

myExcel.SetCellValue(4, columnaActual, "Apron T1");
for (i = 0; i < diferencias.Count(); i++)
{
    if (diferencias[i][2] == 31)
    {
        data.Rows.Add(diferencias[i][0], diferencias[i][1]);
    }
}
myExcel.ImportDataTable(5, columnaActual, data, true);
data.Rows.Clear();
columnaActual = columnaActual + 2;

myExcel.SetCellValue(4, columnaActual, "Apron T2");
for (i = 0; i < diferencias.Count(); i++)
{
    if (diferencias[i][2] == 32)
    {
        data.Rows.Add(diferencias[i][0], diferencias[i][1]);
    }
}
myExcel.ImportDataTable(5, columnaActual, data, true);
data.Rows.Clear();
columnaActual = columnaActual + 2;

myExcel.SetCellValue(4, columnaActual, "Stands T1");
for (i = 0; i < diferencias.Count(); i++)
{
    if (diferencias[i][2] == 21)
    {
        data.Rows.Add(diferencias[i][0], diferencias[i][1]);
    }
}
myExcel.ImportDataTable(5, columnaActual, data, true);
data.Rows.Clear();
columnaActual = columnaActual + 2;

myExcel.SetCellValue(4, columnaActual, "Stands T2");
for (i = 0; i < diferencias.Count(); i++)
{
    if (diferencias[i][2] == 22)
    {
        data.Rows.Add(diferencias[i][0], diferencias[i][1]);
```

```
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "Type 4");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 01)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "Type 5");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 02)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        myExcel.SetCellValue(4, columnaActual, "Rest");
        for (i = 0; i < diferencias.Count(); i++)
        {
            if (diferencias[i][2] == 0)
            {
                data.Rows.Add(diferencias[i][0], diferencias[i][1]);
            }
        }
        myExcel.ImportDataTable(5, columnaActual, data, true);
        data.Rows.Clear();
        columnaActual = columnaActual + 2;

        m yExcel.AutoFitColumn(1, columnaActual - 1, 20); //Adapta el tamaño
de las celdas (columnas de 1 a 2, maximo grosor de 20)
        return myExcel;
    }

    private void SaveScatterButton_Click(object sender, EventArgs e)
//Guarda la tabla de diferencias a excel para poder hacer el Scatter
    {
```

```
        string saveName = nombreFichero;//Añadimos caracteristicas actuales
al nombre del fichero a guardar
        saveFileDialogExcel.FileName = nombreFichero + "_Scatter";

        saveFileDialogExcel.Filter = "File documents (.xlsx)|*.xlsx";
        saveFileDialogExcel.ShowDialog();
        try
        {
            if (this.saveFileDialogExcel.FileName.Equals("") == false)
            {
                if (this.saveFileDialogExcel.FileName.Contains(@"\"))//Controla si
se ha escogido un directorio o se ha cancelado/cerrado el save dialog
                {
                    SLDocument myExcel = SaveScatter();
                    Cursor.Current = Cursors.WaitCursor;
                    myExcel.SaveAs(saveFileDialogExcel.FileName);
                }
            }
            else CrearFormInformativa("Please write a", "name first.", 187, 30, 27,
41);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    private SLDocument SaveSummaryParameter()//prepara el excel para ser
exportado; Exporta solo una tabla (un parametro)
    {
        SLDocument myExcel = new SLDocument();
        myExcel.SetCellValue("A1", "File: " + nombreFichero);//Nombre fichero

        string avionesDetectados = "";
        if (myList.AircraftDetected() != 0)
            avionesDetectados = " from " + myList.AircraftDetected() + " different
transponders ";//Esta frase solo aparece si hay MLAT
        myExcel.SetCellValue("A2", myList.GetNumList() + " packages
avaluated" + avionesDetectados + "between " +
myList.GetPlanI(0).ConvertUTC("SinDecimas") +
                    " hours and " + myList.GetPlanI(myList.GetNumList() -
1).ConvertUTC("SinDecimas") + " hours");//Información del fichero

        myExcel.SetCellValue("A3", Convert.ToString(ParametersGrid[0,
ParametersGrid.CurrentRow.Index].Value) + ":");//Titulo de la grid

        DataTable data = new DataTable();
        data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
```

```
        for (int j = 0; j < SummaryGrid.ColumnCount; j++)//Copiamos filas y
columnas de Summary Grid a data table
        {
            data.Columns.Add(SummaryGrid.Columns[j].Name,
typeof(double));//La informacion que contienen estas columnas es de tipo
double
        }

        int parameter = ParametersGrid.CurrentRow.Index; //A partir de aqui el
codigo diferencia entre cada parametro, lo anterior es generico
        if (!DGPSresults)
        {
            if (parameter >= 1)
            {
                parameter++;//saltamos PA en caso de que no sea tabla DGPS
            }
            for (int i = 0; i < SummaryGrid.RowCount; i++)
            {
                data.Rows.Add(SummaryGrid.Rows[i].HeaderCell.Value,
SummaryGrid[0, i].Value, SummaryGrid[1, i].Value, SummaryGrid[2, i].Value,
SummaryGrid[3, i].Value);
            }
            myExcel.ImportDataTable(4, 1, data, true);//Inserta la tabla en la
posicion i,j=4,1 (la primera celda en excel es la 1,1 ó A1
        }
        else
        {
            if (parameter == 1)
            {
                for (int i = 0; i < SummaryGrid.RowCount; i++)
                {
                    data.Rows.Add(SummaryGrid.Rows[i].HeaderCell.Value,
SummaryGrid[0, i].Value, SummaryGrid[1, i].Value, SummaryGrid[2, i].Value,
SummaryGrid[3, i].Value, SummaryGrid[4, i].Value, SummaryGrid[5, i].Value,
SummaryGrid[6, i].Value);
                }
                myExcel.ImportDataTable(4, 1, data, true);//Inserta la tabla en la
posicion i,j=4,1 (la primera celda en excel es la 1,1 ó A17
            }
            else
            {
                for (int i = 0; i < SummaryGrid.RowCount; i++)
                {
                    data.Rows.Add(SummaryGrid.Rows[i].HeaderCell.Value,
SummaryGrid[0, i].Value, SummaryGrid[1, i].Value, SummaryGrid[2, i].Value,
SummaryGrid[3, i].Value);
                }
                myExcel.ImportDataTable(4, 1, data, true);//Inserta la tabla en la
posicion i,j=4,1 (la primera celda en excel es la 1,1 ó A1
            }
```

```
        }

        SLStyle style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
        int posInicial = 5;//5 por posicion del nombre de fichero, titulo parametro,
informacion .ast, ademas de que excel empieza a enumerar en 1
        for (int i = posInicial; i < SummaryGrid.RowCount + posInicial; i++)
        {
            if (parameter == 0) //UR
            {
                if ((i == 0 + posInicial) || (i == 5 + posInicial) || (i == 8 + posInicial) ||
(i == 11 + posInicial))
                {
                    if (myExcel.GetCellValueAsString(i, 4) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 4) >=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);

                        }
                        myExcel.SetCellStyle(i, 4, style);
                    }
                }
            }
            if (parameter == 1) //Pos Acc
            {
                if (i == 0 + posInicial)
                {
                    if (myExcel.GetCellValueAsString(i, 2) != "")//AsDouble no
acepta null
                    {
                        if (myExcel.GetCellValueAsDouble(i, 2) <=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {
```

```
style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 2, style);
                }
                if (myExcel.GetCellValueAsString(i, 3) != "")//AsDouble no
acepta null
                {
                    if (myExcel.GetCellValueAsDouble(i, 3) <=
myExcel.GetCellValueAsDouble(i, 6))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 3, style);
                }
            }
            else if (i == 5 + posInicial)
            {
                if (myExcel.GetCellValueAsString(i, 2) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 2) <=
myExcel.GetCellValueAsDouble(i, 5))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 2, style);
                }
                if (myExcel.GetCellValueAsString(i, 3) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 3) <=
myExcel.GetCellValueAsDouble(i, 6))
                    {
```

```
style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 3, style);
                }
            }
            else if (i == 8 + posInicial)
            {
                if (myExcel.GetCellValueAsString(i, 4) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 4) <= 20)//max 20m en
Stand PosAcc
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 4, style);
                }
            }
        }
        else if (parameter == 2) //ProbMLATdetection
        {
            if ((i == 0 + posInicial) || (i == 7 + posInicial))
            {
                if (myExcel.GetCellValueAsString(i, 4) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 4) >=
myExcel.GetCellValueAsDouble(i, 5))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {
```

```
style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 4, style);
                    }
                }
            }
            else if (parameter == 3) //PID
            {
                if (i == 15 + posInicial)
                {
                    if (myExcel.GetCellValueAsString(i, 4) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 4) >=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 4, style);
                    }
                }
            }
            else if (parameter == 4) //PFD
            {
                if (i == 14 + posInicial)
                {
                    if (myExcel.GetCellValueAsString(i, 4) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 4) <=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
```

```
                    myExcel.SetCellStyle(i, 4, style);
                }
            }
        }
        else if (parameter == 5) //PFI
        {
            if (i == 15 + posInicial)
            {
                if (myExcel.GetCellValueAsString(i, 4) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 4) <=
myExcel.GetCellValueAsDouble(i, 5))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 4, style);
                }
            }
        }
    }

        myExcel.AutoFitColumn(1, data.Columns.Count, 20);//Adapta el tamaño
de las celdas (columnas de 1 a 5, maximo grosor de 20)
        return myExcel;
    }

    private void buttonSaveSummary_Click(object sender, EventArgs
e)//Exporta el parameter grid actual a excel
    {
        string saveName = nombreFichero;//Añadimos caracteristicas actuales
al nombre del fichero a guardar
        if (DGPSresults)
            saveName = saveName + "_DGPS";
        else saveName = saveName + "_MLAT";
        saveFileDialogExcel.FileName = saveName + "_" +
Convert.ToString(ParametersGrid[0,
ParametersGrid.CurrentRow.Index].Value);

        saveFileDialogExcel.Filter = "File documents (.xlsx)|*.xlsx";
        saveFileDialogExcel.ShowDialog();
        try
        {
```

```
            if (this.saveFileDialogExcel.FileName.Equals("") == false)
            {
                if (this.saveFileDialogExcel.FileName.Contains(@"\"))//Controla si
se ha escogido un directorio o se ha cancelado/cerrado el save dialog
                {
                    SLDocument myExcel = SaveSummaryParameter();
                    Cursor.Current = Cursors.WaitCursor;
                    myExcel.SaveAs(saveFileDialogExcel.FileName);
                }
            }
            else CrearFormInformativa("Please write a", "name first.", 187, 30, 27,
41);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
    }

    private SLDocument SaveSummaryAll(Avaluador avaluador)//Prepara el
excel para ser exportado; Exporta todos los parametros juntos
    {
        SLDocument myExcel = new SLDocument();
        myExcel.SetCellValue("A1", "File: " + nombreFichero);//Nombre fichero
        string avionesDetectados = "";
        if (myList.AircraftDetected() != 0)
            avionesDetectados = " from " + myList.AircraftDetected() + " different
transponders ";//Esta frase solo aparece si hay MLAT
        myExcel.SetCellValue("A2", myList.GetNumList() + " packages
avaluated" + avionesDetectados + "between " +
myList.GetPlanI(0).ConvertUTC("SinDecimas") +
                    " hours and " + myList.GetPlanI(myList.GetNumList() -
1).ConvertUTC("SinDecimas") + " hours");//Información del fichero
        myExcel.SetCellValue("A3", "Summary:");

        i nt lineaActual = 5;//Guarda el valor de la linea en la que estamos
añadiendo información (3 + 1(espacio))
        int parametroActual = 0;

        ////////////////////UR//////////////////////
        myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) + ":");//UR
        lineaActual++;

        DataTable data = new DataTable();
        DataGridView grid = new DataGridView();
        CrearTablaSummary(grid, avaluador);//UR
        data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
```

```
        for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas de
Summary Grid a data table
        {
            data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
        }
        for (int i = 0; i < grid.RowCount; i++)
        {
            data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value, grid[1,
i].Value, grid[2, i].Value, grid[3, i].Value);
        }
        myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla en
la posicion i,j=5,1 (la primera celda en excel es la 1,1 ó A1
        lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

        SLStyle style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
        for (int i = lineaActual; i < grid.RowCount + lineaActual; i++)
        {
            if ((i == 0 + lineaActual) || (i == 5 + lineaActual) || (i == 8 + lineaActual)
|| (i == 11 + lineaActual))
            {
                if (myExcel.GetCellValueAsString(i, 4) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 4) >=
myExcel.GetCellValueAsDouble(i, 5))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 4, style);
                }
            }
        }
        lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla
        int columnasMaximas = data.Columns.Count; //Compara el numero de
columnas de cada tabla y se queda con el mayor


        ////////////////PositionAccuracy////////////////////
        if (DGPSresults)
        {
```

```
            parametroActual++;
            myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) +
":");//PositionAccuracy
            lineaActual++;

            data = new DataTable();
            grid = new DataGridView();
            CrearTablaPositionAccuracy(grid, avaluador);//PosAcc
            data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
            for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas
de Summary Grid a data table
            {
                data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
            }
            for (int i = 0; i < grid.RowCount; i++)
            {
                data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value,
grid[1, i].Value, grid[2, i].Value, grid[3, i].Value, grid[4, i].Value, grid[5, i].Value,
grid[6, i].Value);
            }
            myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla
en la posicion i,j=lineaActual,1 (la primera celda en excel es la 1,1 ó A1
            lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

            style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
            for (int i = lineaActual; i < grid.RowCount + lineaActual; i++)
            {
                if (i == 0 + lineaActual)
                {
                    if (myExcel.GetCellValueAsString(i, 2) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 2) <=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 2, style);
                    }
```

```
                    if (myExcel.GetCellValueAsString(i, 3) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 3) <=
myExcel.GetCellValueAsDouble(i, 6))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 3, style);
                    }
                }
                else if (i == 5 + lineaActual)
                {
                    if (myExcel.GetCellValueAsString(i, 2) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 2) <=
myExcel.GetCellValueAsDouble(i, 5))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 2, style);
                    }
                    if (myExcel.GetCellValueAsString(i, 3) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 3) <=
myExcel.GetCellValueAsDouble(i, 6))
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {
```

```
style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 3, style);
                    }
                }
                else if (i == 8 + lineaActual)
                {
                    if (myExcel.GetCellValueAsString(i, 4) != "")
                    {
                        if (myExcel.GetCellValueAsDouble(i, 4) <= 20) //max 20m en
Stand PosAcc
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                        }
                        else
                        {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                        }
                        myExcel.SetCellStyle(i, 4, style);
                    }
                }
            }
            lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla

            if (data.Columns.Count > columnasMaximas)//Si las columnas son
mayores que las anteriores, reasigna valor
            {
                columnasMaximas = data.Columns.Count;
            }
        }
        ////////////////////ProbOfMLATDetection//////////////////////
        parametroActual++;
        myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) +
":");//ProbOfMlatDetection
        lineaActual++;

        data = new DataTable();
        grid = new DataGridView();
        CrearTablaProbOfMLATDetection(grid, avaluador);//PD
        data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
```

```
        for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas de
Summary Grid a data table
        {
            data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
        }
        for (int i = 0; i < grid.RowCount; i++)
        {
            data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value, grid[1,
i].Value, grid[2, i].Value, grid[3, i].Value);
        }
        myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla en
la posicion i,j=lineaActual,1 (la primera celda en excel es la 1,1 ó A1
        lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

        s tyle = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
        for (int i = lineaActual; i < grid.RowCount + lineaActual; i++)
        {
            if ((i == 0 + lineaActual) || (i == 7 + lineaActual))
            {
                if (myExcel.GetCellValueAsString(i, 4) != "")
                {
                    if (myExcel.GetCellValueAsDouble(i, 4) >=
myExcel.GetCellValueAsDouble(i, 5))
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                    }
                    else
                    {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                    }
                    myExcel.SetCellStyle(i, 4, style);
                }
            }
        }
        lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla

        if (data.Columns.Count > columnasMaximas)//Si las columnas son
mayores que las anteriores, reasigna valor
        {
            columnasMaximas = data.Columns.Count;
        }

        ////////////////////ProbOfIdentification////////////////////
```

```
            parametroActual++;
            myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) + ":");//PID
            lineaActual++;

            data = new DataTable();
            grid = new DataGridView();
            CrearTablaProbOfIdentification(grid, avaluador);//PID
            data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
            for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas de
Summary Grid a data table
            {
                data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
            }
            for (int i = 0; i < grid.RowCount; i++)
            {
                data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value, grid[1,
i].Value, grid[2, i].Value, grid[3, i].Value);
            }
            myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla en
la posicion i,j=lineaActual,1 (la primera celda en excel es la 1,1 ó A1
            lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

            if (myExcel.GetCellValueAsString(grid.RowCount + lineaActual - 1, 4) !=
"")
            {
                style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
                if (myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1,
4) >= myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1, 5))//-1
porque grid empieza a contar en 0
                {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                }
                else
                {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                }
                myExcel.SetCellStyle(grid.RowCount + lineaActual - 1, 4, style);
            }

            lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla
```

```
        if (data.Columns.Count > columnasMaximas)//Si las columnas son
mayores que las anteriores, reasigna valor
        {
            columnasMaximas = data.Columns.Count;
        }

        ////////////////ProbOfFalseDetection//////////////////
        parametroActual++;
        myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) + ":");//PFD
        lineaActual++;

        data = new DataTable();
        grid = new DataGridView();
        CrearTablaProbOfFalseDetection(grid, avaluador);//PFD
        data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
        for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas de
Summary Grid a data table
        {
            data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
        }
        for (int i = 0; i < grid.RowCount; i++)
        {
            data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value, grid[1,
i].Value, grid[2, i].Value, grid[3, i].Value);
        }
        myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla en
la posicion i,j=lineaActual,1 (la primera celda en excel es la 1,1 ó A1
        lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

        if (myExcel.GetCellValueAsString(grid.RowCount + lineaActual - 1, 4) !=
"")
        {
            style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
            if (myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1,
4) <= myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1, 5))//-1
porque grid empieza a contar en 0
            {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
            }
            else
            {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
```

```
            }
            myExcel.SetCellStyle(grid.RowCount + lineaActual - 1, 4, style);
        }

        lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla

        if (data.Columns.Count > columnasMaximas)//Si las columnas son
mayores que las anteriores, reasigna valor
        {
            columnasMaximas = data.Columns.Count;
        }

        //////////////////ProbOfFalseIdentification////////////////////
        parametroActual++;
        myExcel.SetCellValue(lineaActual, 1,
Convert.ToString(ParametersGrid[0, parametroActual].Value) + ":");//PFI
        lineaActual++;

        data = new DataTable();
        grid = new DataGridView();
        CrearTablaProbOfFalseIdentification(grid, avaluador);//PFI
        data.Columns.Add(" ", typeof(string)); //Columna de zonas de
movimiento (sin titulo)
        for (int j = 0; j < grid.ColumnCount; j++)//Copiamos filas y columnas de
Summary Grid a data table
        {
            data.Columns.Add(grid.Columns[j].Name, typeof(double));//La
informacion que contienen estas columnas es de tipo double
        }
        for (int i = 0; i < grid.RowCount; i++)
        {
            data.Rows.Add(grid.Rows[i].HeaderCell.Value, grid[0, i].Value, grid[1,
i].Value, grid[2, i].Value, grid[3, i].Value);
        }
        myExcel.ImportDataTable(lineaActual, 1, data, true);//Inserta la tabla en
la posicion i,j=lineaActual,1 (la primera celda en excel es la 1,1 ó A1
        lineaActual++;//sumamos 1 para usarlo en syle; despues sumamos el
tamaño de la tabla(-1 que ya sumamos ahora) + 2 espacios entre tabla y tabla

        if (myExcel.GetCellValueAsString(grid.RowCount + lineaActual - 1, 4) !=
"")
        {
            style = myExcel.CreateStyle();//Pintamos el fondo verde o rojo
            if (myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1,
4) <= myExcel.GetCellValueAsDouble(grid.RowCount + lineaActual - 1, 5))//-1
porque grid empieza a contar en 0
            {
```

```
style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Green, Color.Transparent);
                }
                else
                {

style.Fill.SetPattern(DocumentFormat.OpenXml.Spreadsheet.PatternValues.Sol
id, Color.Red, Color.Transparent);
                }
                myExcel.SetCellStyle(grid.RowCount + lineaActual - 1, 4, style);
            }

            lineaActual = lineaActual + grid.RowCount + 2; //dejamos 2 espacios
entre tabla y tabla

            if (data.Columns.Count > columnasMaximas)//Si las columnas son
mayores que las anteriores, reasigna valor
            {
                columnasMaximas = data.Columns.Count;
            }

            / ////////////////////////////////
            myExcel.AutoFitColumn(1, columnasMaximas, 20);//Adapta el tamaño
de las celdas (columnas de 1 a 5, maximo grosor de 20)
            return myExcel;
        }

        private void buttonSaveAllSummary_Click(object sender, EventArgs e)
//Boton de guardar todos los parametros a excel
        {
            string saveName = nombreFichero;//Añadimos caracteristicas actuales
al nombre del fichero a guardar
            if (DGPSresults)
                saveName = saveName + "_DGPS";
            else saveName = saveName + "_MLAT";

            saveFileDialogExcel.FileName = saveName + "_Summary";
            saveFileDialogExcel.Filter = "File documents (.xlsx)|*.xlsx";
            saveFileDialogExcel.ShowDialog();
            try
            {
                if (this.saveFileDialogExcel.FileName.Equals("") == false)
                {
                    if (this.saveFileDialogExcel.FileName.Contains(@"\"))//Controla si
se ha escogido un directorio o se ha cancelado/cerrado el save dialog
                    {
                        SLDocument myExcel;
                        if (DGPSresults)
                        {
```

```
                    myExcel = SaveSummaryAll(myAvaluadorDGPS);
                }
                else  myExcel = SaveSummaryAll(myAvaluador);
                Cursor.Current = Cursors.WaitCursor;
                myExcel.SaveAs(saveFileDialogExcel.FileName);
                saveAllCorrectoPorSeparado[1] = true;
                if (saveAllCorrectoPorSeparado[0])//Si el del mapa tambien es
true, el general tambien pasa a ser true
                    saveAllCorrecto = true;
            }
        }
        else CrearFormInformativa("Please write a", "name first.", 187, 30, 27,
41);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
        return;
    }
}

    private void ButtonDerechaSummay_Click(object sender, EventArgs
e)//Botones para pasar de parametro en la summay grid (derecha)
    {
        int i = ParametersGrid.CurrentRow.Index;
        ParametersGrid[0, i].Selected = false;
        if (i == ParametersGrid.RowCount - 1)//De la ultima a la primera
        {
            i = 0;
        }
        else i++;
        ParametersGrid[0, i].Selected = true;
        LabelTituloSummary.Text = Convert.ToString(ParametersGrid[0,
i].Value) + ":";
        SummaryGrid.Rows.Clear();
        SummaryGrid.Columns.Clear();
        if (DGPSresults)
        {
            if (i == 0)
                CrearTablaSummary(this.SummaryGrid, myAvaluadorDGPS);
            else if (i == 1)
                CrearTablaPositionAccuracy(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 2)
                CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 3)
                CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 4)
```

```
                CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 5)
                CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluadorDGPS);
        }
        else
        {
            if (i == 0)
                CrearTablaSummary(this.SummaryGrid, myAvaluador);
            else if (i == 1)
                CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluador);
            else if (i == 2)
                CrearTablaProbOfIdentification(this.SummaryGrid, myAvaluador);
            else if (i == 3)
                CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluador);
            else if (i == 4)
                CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluador);
        }
    }

    private void ButtonIzquierdaSummary_Click(object sender, EventArgs
e)//Botones para pasar de parametro en la summay grid (izquierda)
    {
        int i = ParametersGrid.CurrentRow.Index;
        ParametersGrid[0, i].Selected = false;
        if (i == 0)//De la primera a la ultima
        {
            i = ParametersGrid.RowCount - 1;
        }
        else i--;
        ParametersGrid[0, i].Selected = true;
        LabelTituloSummary.Text = Convert.ToString(ParametersGrid[0,
i].Value) + ":";
        SummaryGrid.Rows.Clear();
        SummaryGrid.Columns.Clear();

        if (DGPSresults)
        {
            if (i == 0)
                CrearTablaSummary(this.SummaryGrid, myAvaluadorDGPS);
            else if (i == 1)
                CrearTablaPositionAccuracy(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 2)
                CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluadorDGPS);
```

```
            else if (i == 3)
                CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 4)
                CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluadorDGPS);
            else if (i == 5)
                CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluadorDGPS);
        }
        else
        {
            if (i == 0)
                CrearTablaSummary(this.SummaryGrid, myAvaluador);
            else if (i == 1)
                CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluador);
            else if (i == 2)
                CrearTablaProbOfIdentification(this.SummaryGrid, myAvaluador);
            else if (i == 3)
                CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluador);
            else if (i == 4)
                CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluador);
        }
    }

    private void ParametersGrid_SelectionChanged(object sender, EventArgs
e)//Si bajas, subes con teclado
    {
        if (ParametersGrid.SelectedRows.Count != 0)
        {
            int i = ParametersGrid.SelectedRows[0].Index;
            if (ParametersGrid[0, 0].Value == null)//En el inicio entra aqui antes
de asignar el nombre a cada row
            {
                ParametersGrid[0, 0].Value = "Update Rate";
            }
            LabelTituloSummary.Text = Convert.ToString(ParametersGrid[0,
i].Value) + ":";
            SummaryGrid.Rows.Clear();
            SummaryGrid.Columns.Clear();
            if (DGPSresults)
            {
                if (i == 0)
                    CrearTablaSummary(this.SummaryGrid, myAvaluadorDGPS);
                else if (i == 1)
                    CrearTablaPositionAccuracy(this.SummaryGrid,
myAvaluadorDGPS);
```

```
                else if (i == 2)
                    CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 3)
                    CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 4)
                    CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluadorDGPS);
                else if (i == 5)
                    CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluadorDGPS);
            }
            else
            {
                if (i == 0)
                    CrearTablaSummary(this.SummaryGrid, myAvaluador);
                else if (i == 1)
                    CrearTablaProbOfMLATDetection(this.SummaryGrid,
myAvaluador);
                else if (i == 2)
                    CrearTablaProbOfIdentification(this.SummaryGrid,
myAvaluador);
                else if (i == 3)
                    CrearTablaProbOfFalseDetection(this.SummaryGrid,
myAvaluador);
                else if (i == 4)
                    CrearTablaProbOfFalseIdentification(this.SummaryGrid,
myAvaluador);
            }
        }
    }

    private void CrearTablaDescartados(DataGridView grid)  //tabla de
vehiculos descartados
    {
        List<string> descartados =
myListConDescartados.LeerVehiculosSquitter();
        grid.RowCount = descartados.Count();
        grid.ColumnCount = 1;
        grid.ColumnHeadersVisible = true;
        grid.RowHeadersVisible = false;
        grid.Columns[0].SortMode =
DataGridViewColumnSortMode.Programmatic;
        grid.Columns[0].Name = "Discarded";
        for (int n = 0; n < grid.RowCount; n++)
        {
            grid[0, n].Style.BackColor = Color.White;
        }
        int j;
```

```
            bool encontrado;
            if (descartados[0] != "Sin Vehiculos")
            {
                for (int i = 0; i < descartados.Count(); i++)
                {
                    grid[0, i].Value = descartados[i];
                    j = 0;
                    encontrado = false;
                    while ((j < myListConDescartados.GetNumList()) && (!encontrado))
                    {
                        if (myListConDescartados.GetPlanI(j).GetICAOAdress() ==
descartados[i])
                        {
                            encontrado = true;
                            grid[0, i].Style.BackColor = Color.Green;
                        }
                        j++;
                    }
                }
            }
            else
            {
                grid[0, 0].Value = "No Vehicles";
            }
        }

        private void CrearTablaNoDescartados(DataGridView grid)  //tabla de
vehiculos no descartados
        {
            double[,] updateRateU = myAvaluador.GetUpdatesUnitarios();
            double[,] diferencia = new double[aircraftDetected.Count(), 3];
//Contiene diferencia (expected-updates reales); UR; indice para Icao
            for (int i = 0; i < updateRateU.GetLength(0); i++)
            {
                diferencia[i, 0] = updateRateU[i, 1] - updateRateU[i, 0];
                if (updateRateU[i, 1] != 0)
                {
                    diferencia[i, 1] = updateRateU[i, 0] / updateRateU[i, 1];
                }
                diferencia[i, 2] = updateRateU[i, 2];
            }
            myAvaluador.OrdenarDiferencias(diferencia, "Mm");

            grid.RowCount = diferencia.GetLength(0);
            grid.ColumnCount = 3;
            grid.ColumnHeadersVisible = true;
            grid.RowHeadersVisible = false;
            grid.Columns[0].SortMode =
DataGridViewColumnSortMode.Programmatic;
            grid.Columns[0].Name = "Not Discarded";
```

```
            grid.Columns[1].Name = "Expected-Real";
            grid.Columns[2].Name = "UR (%)";
            grid.ClearSelection();
            for (int i = 0; i < grid.RowCount; i++)
            {
                grid[0, i].Value = myList.GetPlanI(Convert.ToInt32(diferencia[i,
2])).GetICAOAdress();
                grid[1, i].Value = Math.Round(diferencia[i, 0]);//Diferencia
                grid[2, i].Value = Math.Round(diferencia[i, 1], 3) * 100;//UR
            }
        }

        private void panelMapDescartados_Paint(object sender, PaintEventArgs e)
        {
            try
            {
                Graphics graphics = e.Graphics;
                graphics.DrawImage(myBitmapDescartados, 0, 0,
myBitmapDescartados.Width, myBitmapDescartados.Height);
                graphics.Dispose();
            }
            catch (Exception)
            {
                return;
            }
        }

        private void tablaGridDescartados_CellClick(object sender,
DataGridViewCellEventArgs e)
        {
            int i = e.RowIndex;
            if (i != -1)
            {
                if (Convert.ToString(tablaGridDescartados[0, i].Value) != "No
Vehicles")
                {
                    AddRemoveDiscardButton.Visible = true;
                    ViewWholeButton.Visible = true;
                    AddRemoveDiscardButton.Text = "Remove";
                    tablaGridNoDescartados.ClearSelection();
                    tablaGridDescartados.ClearSelection();
                    tablaGridDescartados[0, i].Selected = true;
                    string icao = Convert.ToString(tablaGridDescartados[0, i].Value);
                    myBitmapDescartados = new
Bitmap(this.panelMapDescartados.Width, this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                    dimensionesDescartado = myLmap.GetDIMENSIONES();
                    proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
```

```
                ViewWholeButton.Text = "View Whole";
                PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
                PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
                panelMapDescartados.Invalidate();
            }
        }
    }

    private void tablaGridDescartados_SelectionChanged(object sender,
EventArgs e)//Si bajas con el teclado por la tabla, actualiza el mapa
    {
        if (tablaGridDescartados.SelectedRows.Count != 0)
        {
            if (Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.SelectedRows[0].Index].Value) != "No Vehicles")
            {
                string icao = Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.SelectedRows[0].Index].Value);
                myBitmapDescartados = new
Bitmap(this.panelMapDescartados.Width, this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                dimensionesDescartado = myLmap.GetDIMENSIONES();
                proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
                ViewWholeButton.Text = "View Whole";
                PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
                PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
                panelMapDescartados.Invalidate();
            }
        }
    }

    private void tablaGridNoDescartados_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        int i = e.RowIndex;
        if (i != -1)
        {
            AddRemoveDiscardButton.Visible = true;
            ViewWholeButton.Visible = true;
            AddRemoveDiscardButton.Text = "Add";
            tablaGridDescartados.ClearSelection();
            tablaGridNoDescartados.ClearSelection();
            tablaGridNoDescartados[0, i].Selected = true;
            string icao = Convert.ToString(tablaGridNoDescartados[0, i].Value);
```

```
            myBitmapDescartados = new
Bitmap(this.panelMapDescartados.Width, this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            dimensionesDescartado = myLmap.GetDIMENSIONES();
            proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
            ViewWholeButton.Text = "View Whole";
            PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
            PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
            panelMapDescartados.Invalidate();
        }
    }

    private void tablaGridNoDescartados_SelectionChanged(object sender,
EventArgs e)//Si bajas con el teclado por la tabla, actualiza el mapa
    {
        if (tablaGridNoDescartados.SelectedRows.Count != 0)
        {
            string icao = Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.SelectedRows[0].Index].Value);
            myBitmapDescartados = new
Bitmap(this.panelMapDescartados.Width, this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            dimensionesDescartado = myLmap.GetDIMENSIONES();
            proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
            ViewWholeButton.Text = "View Whole";
            PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
            PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
            panelMapDescartados.Invalidate();
        }
    }

    private void AddRemoveDiscardButton_Click(object sender, EventArgs e)
//Añade o descarta un vehiculo de la lista de descartados
    {
        List<string> descartados =
myListConDescartados.LeerVehiculosSquitter();
        bool descartado = false; //Hay caminos que no descartan/ añaden un
vehiculo, se evita asi el Reinicio de la form
        if (AddRemoveDiscardButton.Text=="Add")
        {
            if (myListDGPS.GetNumList() != 0)
            {
```

```
            if (myListDGPS.GetPlanI(0).GetICAOAdress() !=
Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.CurrentRow.Index].Value)) //Si hay DGPS no se
puede hacer Add de ese ICAO Address.
            {
                descartados.Add(Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.CurrentRow.Index].Value));
                descartados.Remove("Sin Vehiculos");
                File.Delete("VehiculosAEliminar.txt");
                File.WriteAllLines("VehiculosAEliminar.txt", descartados);
                descartado = true;
            }
            else
            {
                CrearFormInformativa("This vehicle can't be added to the
discarded list because", "it's being used to compare MLAT with D-GPS data.",
508, 190, 27, 51);
            }
        }
        else
        {
            descartados.Add(Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.CurrentRow.Index].Value));
            descartados.Remove("Sin Vehiculos");
            File.Delete("VehiculosAEliminar.txt");
            File.WriteAllLines("VehiculosAEliminar.txt", descartados);
            descartado = true;
        }
    }
    else if (AddRemoveDiscardButton.Text == "Remove")
    {
        if (Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.CurrentRow.Index].Value) != "No Vehicles")
        {
            descartados.Remove(Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.CurrentRow.Index].Value));
        }
        if (descartados.Count()==0)
        {
            descartados.Add("Sin Vehiculos");
        }
        File.Delete("VehiculosAEliminar.txt");
        File.WriteAllLines("VehiculosAEliminar.txt",descartados);
        descartado = true;
    }
    if (descartado)
    {
        RecargarMainForm();
    }
}
```

```
        private void ViewWholeButton_Click(object sender, EventArgs e) //A veces
no se ve una trayectoria en el mapa porque esta en airborne y no pasa por
LEBL. Este boton reajusta el tamaño del mapa
        {
            string icao = "";
            if (AddRemoveDiscardButton.Text == "Add")
            {
                icao = Convert.ToString(tablaGridNoDescartados[0,
tablaGridNoDescartados.CurrentRow.Index].Value);
            }
            else if (AddRemoveDiscardButton.Text == "Remove")
            {
                icao = Convert.ToString(tablaGridDescartados[0,
tablaGridDescartados.CurrentRow.Index].Value);
            }
            if (ViewWholeButton.Text == "View Whole")
            {
                ViewWholeButton.Text = "View Normal";
                double[,] dimensionesVehiculo = myList.GetDimensiones(icao);
                double[,] dimensionesMapa = myLmap.GetDIMENSIONES();
                double[,] dimensionesFinales = dimensionesVehiculo;
                if (dimensionesVehiculo[0, 0] > dimensionesMapa[0, 0])
                {
                    dimensionesFinales[0, 0] = dimensionesMapa[0, 0];
                }
                if (dimensionesVehiculo[0, 1] < dimensionesMapa[0, 1])
                {
                    dimensionesFinales[0, 1] = dimensionesMapa[0, 1];
                }
                if (dimensionesVehiculo[1, 0] > dimensionesMapa[1, 0])
                {
                    dimensionesFinales[1, 0] = dimensionesMapa[1, 0];
                }
                if (dimensionesVehiculo[1, 1] < dimensionesMapa[1, 1])
                {
                    dimensionesFinales[1, 1] = dimensionesMapa[1, 1];
                }
                myBitmapDescartados = new
Bitmap(this.panelMapDescartados.Width, this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
                dimensionesDescartado = dimensionesFinales;
                proporcionDescartados =
myList.GetProporcion(dimensionesFinales,this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
                PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
                PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
                panelMapDescartados.Invalidate();
```

```
            }
            else
            {
                ViewWholeButton.Text = "View Whole";
                dimensionesDescartado = myLmap.GetDIMENSIONES();
                proporcionDescartados =
myLmap.GetPROPORCION(this.panelMapDescartados.Height,
this.panelMapDescartados.Width);
                PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height);
                PintarAllDescartados(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height, icao);
                panelMapDescartados.Invalidate();
            }

        }

        //Menu tool strips
        private void loadFileToolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (saveAllCorrecto)
            {
                Form1 nuevoFichero = new Form1();
                nuevoFichero.StartPosition = FormStartPosition.CenterScreen;
                nuevoFichero.SetNuevoFichero(true);
                nuevoFichero.Show();
                if (nuevoFichero.CerrarMain())
                {
                    this.Close();
                }
            }
            else
            {
                bool exit = false; //Controla si se quiere salir en la Form Close
                bool save = false; //Controla si se quiere guardar el file
                FormClose close = new FormClose();
                close.StartPosition = FormStartPosition.CenterScreen;
                close.ShowDialog();
                exit = close.ExportarExit();
                save = close.ExportarSave();
                if (exit)
                {
                    if (save) //Guarda todos los documentos en una carpeta
                    {
                        saveAllToolStripMenuItem.PerformClick();
                        if (saveAllCorrecto)
                        {
                            CrearFormInformativa("File correctly saved,", "choose a new
file.", 231, 54, 27, 34);
                            loadFileToolStripMenuItem.PerformClick();
```

```
                }
            }
            else
            {
                saveAllCorrecto = true;
                loadFileToolStripMenuItem.PerformClick();
            }
        }
    }
}

private void saveAllToolStripMenuItem_Click(object sender, EventArgs e)
{
    bool origenDescartados = false;
    if (mapaDescartados)//Miramos si esta en la ventana de descartados
    {
        origenDescartados = true;
    }
    mapaDescartados = false;//Lo ponemos false para que no guarde el
formato basico, despues, segun el origen, volverá a ser true

    Bitmap saveBitmap;
    SLDocument myExcel;
    saveFileDialogAll.FileName = nombreFichero + "_Results";
    saveFileDialogAll.ShowDialog();
    try
    {
        if (this.saveFileDialogAll.FileName.Equals("") == false)
        {
            if (this.saveFileDialogAll.FileName.Contains(@"\"))//Controla si se
ha escogido un directorio o se ha cancelado/cerrado el save dialog
            {
                Cursor.Current = Cursors.WaitCursor;
                Directory.CreateDirectory(saveFileDialogAll.FileName);

                bool saveDGPS = DGPS;//Guardamos el valor actual de los
boleanos
                bool saveMLATandDGPS = MLATandDGPS;
                bool saveSegmentation = segmentation;
                bool saveZoom = zoom;
                bool saveHideBack = HideBack;
                bool saveDGPSresults = DGPSresults;
                int indiceParametro = ParametersGrid.CurrentRow.Index;

                ResetearBoleanos();//Lo pone todo en false
                saveBitmap = SaveMap();
                saveBitmap.Save(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_MLAT_Map.png"));
                segmentation = true;
                saveBitmap = SaveMap();
```

```
                saveBitmap.Save(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_MLAT_Segmented_Map.png"));
                ResetearBoleanos();
                HideBack = true;
                zoom = true;
                saveBitmap = SaveMap();
                saveBitmap.Save(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_MLAT_ZoomOut_WithOutBackground_Map.png"));
                if (myListDGPS.GetNumList() != 0)
                {
                    ResetearBoleanos();
                    DGPS = true;
                    saveBitmap = SaveMap();
                    saveBitmap.Save(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_DGPS_Map.png"));
                    ResetearBoleanos();
                    MLATandDGPS = true;
                    HideBack = true;
                    saveBitmap = SaveMap();
                    saveBitmap.Save(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_MLATandDGPS_WithOutBackground_Map.png"));

                    ResetearBoleanos();
                    DGPSresults = true;
                    CrearTablaParameters();
                    myExcel = SaveSummaryAll(myAvaluadorDGPS);
                    myExcel.SaveAs(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_DGPS_Summary.xlsx"));
                }

                ResetearBoleanos();
                CrearTablaParameters();
                myExcel = SaveSummaryAll(myAvaluador);
                myExcel.SaveAs(Path.Combine(saveFileDialogAll.FileName,
nombreFichero + "_MLAT_Summary.xlsx"));

                saveAllCorrecto = true;

                DGPS = saveDGPS;//Volvemos a asignar los valores originales
de los boleanos
                MLATandDGPS = saveMLATandDGPS;
                segmentation = saveSegmentation;
                zoom = saveZoom;
                HideBack = saveHideBack;
                DGPSresults = saveDGPSresults;
                CrearTablaParameters();
                ParametersGrid.ClearSelection();
                ParametersGrid[0, indiceParametro].Selected = true;
                LabelTituloSummary.Text = Convert.ToString(ParametersGrid[0,
indiceParametro].Value) + ":";
```

```
                }
                else
                {
                    saveAllCorrecto = false;
                }

            }
            else
            {
                CrearFormInformativa("Please write a", "name first.", 187, 30, 27,
41);
                saveAllCorrecto = false;
            }

            if (origenDescartados)//Miramos si esta en la ventana de descartados
y lo devolvemos a ahi
            {
                mapaDescartados = true;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            saveAllCorrecto = false;
            if (origenDescartados)//Miramos si esta en la ventana de descartados
y lo devolvemos a ahi
            {
                mapaDescartados = true;
            }
            return;
        }
    }

    private void mapToolStripMenuItem_Click(object sender, EventArgs e)
    {
        panelContenedorMapa.Visible = true;
        panelContenedorResultados.Visible = false;
        panelContenedorDescartados.Visible = false;
        mapaDescartados = false;
        labelCursor.Text = "Cursor: --.--m, --.--m (X,Y)";
    }

    private void summaryToolStripMenuItem_Click(object sender, EventArgs
e)
    {
        panelContenedorMapa.Visible = false;
        panelContenedorResultados.Visible = true;
        panelContenedorDescartados.Visible = false;
        mapaDescartados = false;
    }
```

```
        private void discardedVehiclesToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            panelContenedorMapa.Visible = false;
            panelContenedorResultados.Visible = false;
            panelContenedorDescartados.Visible = true;
            mapaDescartados = true;
            myBitmapDescartados = new Bitmap(this.panelMapDescartados.Width,
this.panelMapDescartados.Height,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            PintarFondoPantalla(myBitmapDescartados,
this.panelMapDescartados.Width, this.panelMapDescartados.Height); //Genera
el bmp con el mapa de LEBL
            tablaGridDescartados.ClearSelection();
            tablaGridNoDescartados.ClearSelection();
            AddRemoveDiscardButton.Visible = false;
            ViewWholeButton.Visible = false;
        }

        private void closeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void MainForm_FormClosing(object sender,
FormClosingEventArgs e)//Evento que se activa cuando se cierra la Mainform
        {
            if (!saveAllCorrecto)
            {
                e.Cancel = true;
                bool exit = false; //Controla si se quiere salir en la Form Close
                bool save = false; //Controla si se quiere guardar el file
                FormClose close = new FormClose();
                close.StartPosition = FormStartPosition.CenterScreen;
                close.ShowDialog();
                exit = close.ExportarExit();
                save = close.ExportarSave();
                if (exit)
                {
                    if (save) //Guarda todos los documentos en una carpeta
                    {
                        saveAllToolStripMenuItem.PerformClick();
                        if (saveAllCorrecto)
                            e.Cancel = false;
                    }
                    else e.Cancel = false;
                }
                //Si exit=false e.cancel se mantiene true y todo se queda como
estaba
```

```
            }
        }

        //Subfunciones
        public void CrearFormInformativa(string info1, string info2, int formWidth,
int Xbutton, int Xlabel1, int Xlabel2)//Valores puestos a mano para cuadradrar el
texto segun la longitud del propio texto
        {
            FormInformativa formI = new FormInformativa();
            formI.ImportarInformacion(info1, info2, formWidth, Xbutton, Xlabel1,
Xlabel2);
            formI.ShowDialog();
        }
    }
}
```

FormClose.cs

```
using System;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
   public partial class FormClose : Form
   {
      bool exit; //Controla si se quiere salir de la app o no
      bool save; //Controla si se quiere guardar el file

      public FormClose()
      {
         InitializeComponent();
      }

      public bool ExportarExit()
      {
         return exit;
      }

      public bool ExportarSave()
      {
         return save;
      }

      private void DontSaveButton_Click(object sender, EventArgs e)
      {
         exit = true;
         save = false;
         Close();
      }

      private void CancelarButton_Click(object sender, EventArgs e)
      {
         exit = false;
         save = false;
         Close();
      }

      private void SaveButton_Click(object sender, EventArgs e)
      {
         exit = true;
         save = true;
         Close();
      }
   }
}
```

FormInformativa.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
   public partial class FormInformativa : Form
   {
      public FormInformativa()
      {
         InitializeComponent();
      }

      public void ImportarInformacion(string Info1, string Info2, int formWidth, int
xButton, int xLabel1, int xLabel2)
      {
         this.label1.Text = Info1;
         this.label2.Text = Info2;
         this.Size = new Size(formWidth, this.Height);
         this.OkButton.Location = new Point(xButton, OkButton.Location.Y);
         this.label1.Location = new Point(xLabel1, label1.Location.Y);
         this.label2.Location = new Point(xLabel2, label2.Location.Y);
      }

      private void OkButton_Click(object sender, EventArgs e)
      {
         Close();
      }
   }
}
```

FormInformativaYesNo.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
   public partial class FormInformativaYesNo : Form
   {
      public FormInformativaYesNo()
      {
         InitializeComponent();
      }

      bool respuesta = false; //false = Cancel
```

```
        public bool ExportarRespuesta()
        {
            return respuesta;
        }

        public void ImportarInformacion(string Info1, string Info2, int formWidth, int
xButton1, int xButton2, int xLabel1, int xLabel2)
        {
            this.label1.Text = Info1;
            this.label2.Text = Info2;
            this.Size = new Size(formWidth, this.Height);
            this.OkButton.Location = new Point(xButton1, OkButton.Location.Y);
            this.CancelarButton.Location = new Point(xButton2,
CancelarButton.Location.Y);
            this.label1.Location = new Point(xLabel1, label1.Location.Y);
            this.label2.Location = new Point(xLabel2, label2.Location.Y);
        }

        private void OkButton_Click(object sender, EventArgs e)
        {
            respuesta = true;
            Close();
        }

        private void CancelButton_Click(object sender, EventArgs e)
        {
            respuesta = false;
            Close();
        }
    }
}
```

ICAOcode.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
    public partial class ICAOcode : Form
    {
```

```
string Icao;

public ICAOcode()
{
    InitializeComponent();
}

public string ExportarIcao()
{
    return Icao;
}

private void SubmitButton_Click(object sender, EventArgs e)
{
    Color myRgbColor = new Color();
    myRgbColor = Color.FromArgb(255, 63, 63);
    try
    {
        string temporalIcao = textBoxIcao.Text;
        if (temporalIcao.Count()==6)
        {
            if (temporalIcao.Contains(" "))
            {
                textBoxIcao.BackColor = myRgbColor;
                timer1.Interval = 800; // en milisegundos = 1segundo
                timer1.Start();
            }
            else
            {
                Icao = temporalIcao;
                Close();
            }
        }
        else
        {
            textBoxIcao.BackColor = myRgbColor;
            timer1.Interval = 800; // en milisegundos = 1segundo
            timer1.Start();
        }
    }
    catch (FormatException)
    {
        textBoxIcao.BackColor = myRgbColor;
        timer1.Interval = 800; // en milisegundos = 1segundo
        timer1.Start();
        MessageBox.Show("Incorrect data structure");
        return;
    }
}
```

```
        private void ICAOcode_Load(object sender, EventArgs e)
        {
            textBoxIcao.Text = "344399"; //ICAO Address que se suele usar
            textBoxIcao.SelectAll();
            Icao = " ";
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            textBoxIcao.BackColor = Color.White;//vuelve al color de la form
            timer1.Stop();
        }

        private void CancelarButton_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void textBoxIcao_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == Convert.ToChar(Keys.Enter))
            {
                SubmitButton.PerformClick();
            }
        }
    }
}
```

Matches.cs

```
using System;
using System.Windows.Forms;

namespace ED_SMR_MLAT_Performance
{
    public partial class Matches : Form
    {
        int matches;

        public Matches()
        {
            InitializeComponent();
        }

        public void ImportarMatches(int Match)
        {
            this.matches = Match;
        }
```

```
        private void SaveButton_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void Matches_Load(object sender, EventArgs e)
        {
            if (matches!=0)
            {
                labelTitle.Text = "File correctly added!";
                labelSubTitle.Text = matches + " points of D-GPS matched\nwith
MLAT";
            }
            else
            {
                labelTitle.Text = "Something happened...";
                labelSubTitle.Text = "0 points of D-GPS matched\nwith MLAT";
            }
        }
    }
}
```

app.manifest

```
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-
com:asm.v1">
  <assemblyIdentity version="1.0.0.0" name="MyApplication.app"/>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
    <security>
      <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
        <!-- Opciones del manifiesto UAC
             Si quiere cambiar el nivel del Control de cuentas de usuario de
Windows reemplace el
             nodo requestedExecutionLevel por uno de los siguientes.

        <requestedExecutionLevel  level="asInvoker" uiAccess="false" />
        <requestedExecutionLevel  level="requireAdministrator" uiAccess="false"
/>
        <requestedExecutionLevel  level="highestAvailable" uiAccess="false" />

            Especificar el elemento requestedExecutionLevel deshabilitará la
virtualización de archivos y registros.
            Quite este elemento si la aplicación necesita esta virtualización para la
compatibilidad
            con versiones anteriores.

        -->
```

```
      <requestedExecutionLevel  level="requireAdministrator" uiAccess="false"
/>
    </requestedPrivileges>
  </security>
 </trustInfo>

 <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
  <application>
   <!-- Una lista de las versiones de Windows en la que se ha probado esta
aplicación y
        con la que se ha diseñado para que trabaje. Quite la marca de
comentario de los elementos adecuados y Windows seleccionará
        automáticamente el entorno más compatible. -->

   <!-- Windows Vista -->
   <!--<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}" />-->

   <!-- Windows 7 -->
   <!--<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}" />-->

   <!-- Windows 8 -->
   <!--<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}" />-->

   <!-- Windows 8.1 -->
   <!--<supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d0da78}" />-->

   <!-- Windows 10 -->
   <!--<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />-->

  </application>
 </compatibility>

 <!-- Indica que la aplicación tiene reconocimiento de PPP y Windows no la
escalará de forma automática a
     PPP superiores. Las aplicaciones de Windows Presentation Foundation
(WPF) tienen reconocimiento de PPP automático y no necesitan
     participar. Las aplicaciones de Windows Forms que apuntan a .NET
Framework 4.6 que participan en esta configuración, también
     deben establecer la configuración
'EnableWindowsFormsHighDpiAutoResizing' en 'true' en app.config. -->
 <!--
 <application xmlns="urn:schemas-microsoft-com:asm.v3">
  <windowsSettings>
   <dpiAware
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAw
are>
  </windowsSettings>
 </application>
 -->
```

```
  <!-- Habilitar los temas para los controles y cuadros de diálogo comunes de
Windows (Windows XP y versiones posteriores) -->
  <!--
  <dependency>
    <dependentAssembly>
      <assemblyIdentity
          type="win32"
          name="Microsoft.Windows.Common-Controls"
          version="6.0.0.0"
          processorArchitecture="*"
          publicKeyToken="6595b64144ccf1df"
          language="*"
        />
    </dependentAssembly>
  </dependency>
  -->

</assembly>
```

Librerías

DecodificadorMensaje.cs

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Codigo
{
   public class DecodificadorMensaje
   {
      //Atributos

      int CAT; //Categoria: 2 numeros. Ej: 10
      string SIC; //3 numeros (es string porque sino al poner 000 llegaria un 0).
      string SAC; //3 numeros (idem SIC)
      double UTC; //En minutos y decimas de minuto.
      string trackNumber; //4 cifras
      string FL; //X cifras.
      string callSign; //Target Identification
      string ICAOAdress; //Poner en hexadecimal
      double cartX; //Cartesian coordinatex, X component[m]
      double cartY; //[m]
      double polarRho; //Polar coordinates, Rho component[m].
      double polarTheta; //[º]
      double cartFromPolarX; //Función en la libreria que covierte de polar a
cartesiano. En la tabla mostrar info en polar, pero hace falta en
      // cartesiando para plotear. Idem para los siguientes "cartFrom".
      double cartFromPolarY;
      string WGS84Lat; //Latitud en WGS84. Sin signo, hay abajo una variable
para el signo.
      string WGS84Long; //Longitud en WGS84. Sin signo.
      string[] WGS84Sign; //Vector con 2 posiciones. En la primera "N" ó "S"; en
la segunda "E" ó "W". Ej: WGS84Sign=[N,E]
      double MLATfromSMRX; //Hay una funcion que cambia la referencia del
SMR al MLAT
      double MLATfromSMRY;
      double cartTrackVelX; //Cartesian track velocity (X component).
      double cartTrackVelY;  //[m/s]
      double polarTrackVelV; //Polar track velocity (kt).
      double polarTrackVelAngle; //[º].
      double accX; //Calculated Acceleration X component
      double accY;
      int zona; //Diferencia entre cada zona del aeropuerto: runway, stand,
apron, taxi y airborne
      int indiceDGPS; //Para el dichero DGPS, guarda la posicion del vector
myList con el que se tienen que comparar lo de la lista DGPS
      double UTCcorregido; //En minutos y decimas de minuto.
```

```
        //Constructor: asigna los valores nulos
        public DecodificadorMensaje()
        {
            this.CAT = -1;
            this.SIC = "No Data";
            this.SAC = "No Data";
            this.UTC = -1;
            this.trackNumber = "No Data";
            this.FL = null;
            this.callSign = "No Data";
            this.ICAOAdress = "No Data";
            this.cartX = Math.Pow(10, 8); //Valor imposible para CartX y siguientes
double/int que puedan ser negativos;
            this.cartY = Math.Pow(10, 8);
            this.polarRho = Math.Pow(10, 8);
            this.polarTheta = Math.Pow(10, 8);
            this.cartFromPolarX = Math.Pow(10, 8);
            this.cartFromPolarY = Math.Pow(10, 8);
            this.WGS84Lat = null;
            this.WGS84Long = null;
            this.WGS84Sign = null;
            this.MLATfromSMRX = Math.Pow(10, 8);
            this.MLATfromSMRY = Math.Pow(10, 8);
            this.cartTrackVelX = Math.Pow(10, 8);
            this.cartTrackVelY = Math.Pow(10, 8);
            this.polarTrackVelV = Math.Pow(10, 8);
            this.polarTrackVelAngle = Math.Pow(10, 8);
            this.accX= Math.Pow(10, 8);
            this.accY= Math.Pow(10, 8);
            this.zona = -1; //Solo permanece asi si su posicion no es conocida.
            this.indiceDGPS = -1;
            this.UTCcorregido = -1;
        }

        //Funciones principales

        public void GetMlatFromSMR() //Cambia la referencia del SMR al MLAT
para centrar todos los datos en el mismo origen
        {
            string[] Mlat = new string[] { "411749426N", "0020442410E" };
//Coordenadas del MLAT y SMR
            string[] SMR = new string[] { "411744226N", "0020542411E" };
            double[] cartMlat = new double[2];
            double[] cartSMR = new double[2];
            cartMlat[0] = ConvertWGStoRad(Mlat[0], 0);
            cartMlat[1] = ConvertWGStoRad(Mlat[1], 1);
            cartSMR[0] = ConvertWGStoRad(SMR[0], 0);
            cartSMR[1] = ConvertWGStoRad(SMR[1], 1);
            double[] incremento = new double[2];
```

```
            incremento = ConvertRadtoXY(cartMlat, cartSMR[0], cartSMR[1]);
            if ((this.SIC == "007") &&(this.cartX != Math.Pow(10, 8)) && (this.cartY
!= Math.Pow(10, 8)))
            {
                this.MLATfromSMRX = this.cartX + incremento[0];
                this.MLATfromSMRY = this.cartY + incremento[1];
            }
        }

        public int[] GetCartFromWGS84() //int[x,y] Utiliza ConvertWGStoRad y
lego ConvertRadtoXY. Lo referencia al MLAT
        {
            //CAT21 usa ADS-B. Este esta centrado en MLAT
            string[] Mlat = new string[] { "411749426N", "0020442410E" };
//Coordenadas del MLAT y SMR
            double[] cartMlat = new double[2];
            int[] result = new int[2];
            if ((this.WGS84Lat != null) && (this.WGS84Long != null))
            {
                cartMlat[0] = ConvertWGStoRad(Mlat[0], 0);
                cartMlat[1] = ConvertWGStoRad(Mlat[1], 1);
                result[0] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(this.WGS84Lat + this.WGS84Sign[0], 0),
ConvertWGStoRad(this.WGS84Long + this.WGS84Sign[1], 1))[0]);
                result[1] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(this.WGS84Lat + this.WGS84Sign[0], 0),
ConvertWGStoRad(this.WGS84Long + this.WGS84Sign[1], 1))[1]);
                return (result);
            }
            else
            {
                result[0] = Convert.ToInt32(Math.Pow(10, 8));
                result[1] = Convert.ToInt32(Math.Pow(10, 8));
                return (result);
            }
        }

        static string BytesEnString(byte[] buffer, int offset, int count)
        /*Dado un vector de bytes, un offset y un entero count, devuelve un string
de
         * los bytes expresados en binario, tantos bytes como indique count y a
partir del byte en la posicion offset*/
        {
            string byteString;
            string ceros;
            string resultado = "";
            int contador = 0; //Cuenta por qué byte vamos
            int i = 0; //Cuenta los ceros a añadir para el byte actual
            while (contador < count)
            {
```

```
            byteString = Convert.ToString(buffer[offset], 2);
            ceros = "";
            //Añadir ceros
            i = 0;
            while (i < 8 - byteString.Length)
            {
                ceros = ceros + "0";
                i = i + 1;
            }
            //Guardamos el byte en string e incrementamos contador y offset
            resultado = resultado + ceros + byteString;
            offset = offset + 1;
            contador = contador + 1;
        }

        return resultado;
    }

    static int ConvertirCA2aEntero(string CA2)
    /*Convierte un string en Complemento a 2 a número entero decimal (es
decir, suponiendo que LSB tiene valor 1)*/
    /*ADVERTENCIA: El string debe tener al menos dos caracteres, ya que
MSB solo sirve para indicar el signo del numero*/
    {
        //Comprobamos si es un número negativo
        bool negativo;
        if (Convert.ToInt32("" + CA2[0]) == 1)
            negativo = true;
        else
            negativo = false;

        //Lectura del numero y conversion a decimal
        int result;
        string binario = "";
        int i = 1; //Primer bit solo indica el signo
        if (negativo)
        {
            while (i < CA2.Length)
            {
                if (Convert.ToInt32("" + CA2[i]) == 1)
                    binario = binario + "0";
                if (Convert.ToInt32("" + CA2[i]) == 0)
                    binario = binario + "1";
                i = i + 1;
            }
            result = Convert.ToInt32(binario, 2) + 1;
            result = -result;
        }
        else
        {
```

```
            while (i < CA2.Length)
            {
                if (Convert.ToInt32("" + CA2[i]) == 1)
                    binario = binario + "1";
                if (Convert.ToInt32("" + CA2[i]) == 0)
                    binario = binario + "0";
                i = i + 1;
            }
            result = Convert.ToInt32(binario, 2);
        }
        return result;
    }

    static string TraducirCallSign(string caracteresBinario)
    /*Dado un string en binario con los 8 caracteres de un CallSign (6 bits
cada uno) te devuelve el string en ASCII*/
    {
        string callSign = "";      //string con el CallSign descifrado
        string caracter;           //Caracter actual a descifrar
        int caractBit;             //8 caracteres
        int bit = 0;               //6 bits por caracter (b6 b5 b4 b3 b2 b1), en total 48
bits a descifrar
        int parte1;                //Cada caracter se dividirá en dos partes: b6b5
(parte1) y b4b3b2b1 (parte2)
        int parte2;                //Decodificación basada en el International
Alphabet Number 5 (Ver Documento ICAO Anexo 10, Volumen IV, sección
3.1.2.9)
        bool callsignUtil = false;  //Controla si el callsign recibido es legible. Si
solo se reciben caracteres especiales y/o espacios se considera ilegible

        //El string de entrada debe tener un (múltiplo de 6) caracteres
        if ((caracteresBinario.Length) % 6 != 0)
            callSign = "not correctly interpreted";
        else
        {
            while (bit < caracteresBinario.Length)
            {
                //Dividimos el caracter actual en dos partes
                caractBit = 0;
                caracter = "";
                while (caractBit < 2)
                {
                    caracter = caracter + caracteresBinario[bit];
                    caractBit = caractBit + 1;
                    bit = bit + 1;
                }
                parte1 = Convert.ToInt32(caracter, 2);
                caracter = "";
                while (caractBit < 6)
                {
```

```
            caracter = caracter + caracteresBinario[bit];
            caractBit = caractBit + 1;
            bit = bit + 1;
        }
        parte2 = Convert.ToInt32(caracter, 2);

        //Decodificacion del caracter segun IA-5
        if (parte1 == 0)
        {
            if (parte2 == 0)
                callSign = callSign + " ";
            if (parte2 == 1)
            {
                callSign = callSign + "A";
                callsignUtil = true;
            }
            if (parte2 == 2)
            {
                callSign = callSign + "B";
                callsignUtil = true;
            }
            if (parte2 == 3)
            {
                callSign = callSign + "C";
                callsignUtil = true;
            }
            if (parte2 == 4)
            {
                callSign = callSign + "D";
                callsignUtil = true;
            }
            if (parte2 == 5)
            {
                callSign = callSign + "E";
                callsignUtil = true;
            }
            if (parte2 == 6)
            {
                callSign = callSign + "F";
                callsignUtil = true;
            }
            if (parte2 == 7)
            {
                callSign = callSign + "G";
                callsignUtil = true;
            }
            if (parte2 == 8)
            {
                callSign = callSign + "H";
                callsignUtil = true;
```

```
        }
        if (parte2 == 9)
        {
            callSign = callSign + "I";
            callsignUtil = true;
        }
        if (parte2 == 10)
        {
            callSign = callSign + "J";
            callsignUtil = true;
        }
        if (parte2 == 11)
        {
            callSign = callSign + "K";
            callsignUtil = true;
        }
        if (parte2 == 12)
        {
            callSign = callSign + "L";
            callsignUtil = true;
        }
        if (parte2 == 13)
        {
            callSign = callSign + "M";
            callsignUtil = true;
        }
        if (parte2 == 14)
        {
            callSign = callSign + "N";
            callsignUtil = true;
        }
        if (parte2 == 15)
        {
            callSign = callSign + "O";
            callsignUtil = true;
        }
    }
    if (parte1 == 1)
    {
        if (parte2 == 0)
        {
            callSign = callSign + "P";
            callsignUtil = true;
        }
        if (parte2 == 1)
        {
            callSign = callSign + "Q";
            callsignUtil = true;
        }
        if (parte2 == 2)
```

```
        {
            callSign = callSign + "R";
            callsignUtil = true;
        }
        if (parte2 == 3)
        {
            callSign = callSign + "S";
            callsignUtil = true;
        }
        if (parte2 == 4)
        {
            callSign = callSign + "T";
            callsignUtil = true;
        }
        if (parte2 == 5)
        {
            callSign = callSign + "U";
            callsignUtil = true;
        }
        if (parte2 == 6)
        {
            callSign = callSign + "V";
            callsignUtil = true;
        }
        if (parte2 == 7)
        {
            callSign = callSign + "W";
            callsignUtil = true;
        }
        if (parte2 == 8)
        {
            callSign = callSign + "X";
            callsignUtil = true;
        }
        if (parte2 == 9)
        {
            callSign = callSign + "Y";
            callsignUtil = true;
        }
        if (parte2 == 10)
        {
            callSign = callSign + "Z";
            callsignUtil = true;
        }
    }
    if (parte1 == 2)
    {
        if (parte2 == 0)
            //Caracter especial
            callSign = callSign + "";
```

```
}
if (parte1 == 3)
{
    if (parte2 == 0)
    {
        callSign = callSign + "0";
        callsignUtil = true;
    }
    if (parte2 == 1)
    {
        callSign = callSign + "1";
        callsignUtil = true;
    }
    if (parte2 == 2)
    {
        callSign = callSign + "2";
        callsignUtil = true;
    }
    if (parte2 == 3)
    {
        callSign = callSign + "3";
        callsignUtil = true;
    }
    if (parte2 == 4)
    {
        callSign = callSign + "4";
        callsignUtil = true;
    }
    if (parte2 == 5)
    {
        callSign = callSign + "5";
        callsignUtil = true;
    }
    if (parte2 == 6)
    {
        callSign = callSign + "6";
        callsignUtil = true;
    }
    if (parte2 == 7)
    {
        callSign = callSign + "7";
        callsignUtil = true;
    }
    if (parte2 == 8)
    {
        callSign = callSign + "8";
        callsignUtil = true;
    }
    if (parte2 == 9)
    {
```

```
                    callSign = callSign + "9";
                    callsignUtil = true;
                }
            }
        }
        if (!callsignUtil)
            callSign = "No Data";
    }
    return callSign;
}

public void LeerSiguienteMensaje(FileStream fs)
//Decodifica el siguiente mensaje de un archivo Asterix y rellena el
DecodificadorMensaje con su informacion
{
    //Creo un buffer y leo los primeros tres bits
    byte[] buffer = new byte[1024];
    int byteActual = 0; //Contador que lleva por qué byte vamos en el buffer

    //Leo el primer byte con la CAT
    fs.Read(buffer, byteActual, 1);

    //Si el buffer está lleno, rellenamos el mensaje. Estará lleno si el primer
byte es distinto a 0 (CAT!=0)
    if (buffer[byteActual] != 0)
    {
        this.CAT=buffer[byteActual];
        byteActual = byteActual + 1;

        //Segun el tipo de CAT se decodifica de una forma u otra
        if (CAT == 10)
        {
            DecodificarCAT10(fs, buffer, byteActual);
            GetMlatFromSMR();
        }
        else if (CAT == 20)
        {
            DecodificarCAT20(fs, buffer, byteActual);
            GetMlatFromSMR();
        }
        else
        {
            SaltarMensaje(fs, buffer, byteActual);
        }
    }
}

public void SaltarMensaje(FileStream fs, byte[] buffer, int byteActual)
{
```

```
            //No necesario decodificar, pero hace falta leer length para saltar
mensaje
            fs.Read(buffer, byteActual, 2);
            int LEN = Convert.ToInt32(BytesEnString(buffer, 1, 2), 2);
            byteActual = byteActual + 2;
            fs.Read(buffer, byteActual, LEN - 3);
        }

        static int[] BitVector(string stringBits)
        {
            //Convierte un string de 0 y 1 en un vector de int que en cada
componente guarda un dígito
            int[] resultado = new int[stringBits.Length];
            int i = 0;
            while (i < stringBits.Length)
            {
                resultado[i] = Convert.ToInt32(Convert.ToString(stringBits[i]));
                i = i + 1;
            }
            return resultado;
        }

        public void DecodificarCAT10(FileStream fs, byte[] buffer, int byteActual)
        {
            //Leemos la Length del mensaje
            fs.Read(buffer, byteActual, 2);
            int LEN = Convert.ToInt32(BytesEnString(buffer, 1, 2), 2);
            byteActual = byteActual + 2;

            //Guardamos el resto del mensaje en buffer
            fs.Read(buffer, byteActual, LEN - 3);

            //Leemos FSPEC
            string lectura;           //Guarda el byte actual de FSPEC en un string
            int bit;                  //Contador del bit actual que se está leyendo
            bool[] FSPEC = new bool[28];    //el vector indica qué Data Items se han
enviado en el mensaje
            int FRN = 0;                //Indica la posicion actual de FSPEC

            bool FX = true; //Cuando se lee FSPEC, si el último bit es 1 se sigue
leyendo el siguiente byte. Si no, acaba FSPEC.
            while (FX)
            {
                lectura = BytesEnString(buffer, byteActual, 1);

                //Leemos los bits y guardamos qué Data Items se han enviado
                bit = 0;
                while (bit < 7)
                {
                    if (Convert.ToInt32("" + lectura[bit]) == 1)
```

```
                FSPEC[FRN] = true;
            if (Convert.ToInt32("" + lectura[bit]) == 0)
                FSPEC[FRN] = false;

            bit = bit + 1;
            FRN = FRN + 1;
        }
        //leemos último bit
        if (Convert.ToInt32("" + lectura[bit]) == 0)
            FX = false;

        byteActual = byteActual + 1;
    }

    //Lectura de cada uno de los Data Fields

    //FRN=1, Data Item I010/010: Data Source Identifier
    if (FSPEC[0])
    {
        //SAC
        lectura = Convert.ToString(buffer[byteActual], 10);
        while (lectura.Length < 3)
            lectura = "0" + lectura;
        this.SAC= lectura;
        byteActual++;

        //SIC
        lectura = Convert.ToString(buffer[byteActual], 10);
        while (lectura.Length < 3)
            lectura = "0" + lectura;
        this.SIC = lectura;
        byteActual++;
    }

    //FRN=2, Data Item I010/000: Message Type
    if (FSPEC[1])
    {
        byteActual++;
    }

    //FRN=3, Data Item I010/020: Target Report Descriptor (Variable Length
Data)
    if (FSPEC[2])
    {
        //Lectura de los bytes que forman el Data Item (mismo algoritmo que
FSPEC)
        FX = true;
        while (FX)
        {
            lectura = BytesEnString(buffer, byteActual, 1);
```

```
                bit = 0;
                while (bit < 7)
                {
                    bit = bit + 1;
                }
                if (Convert.ToInt32("" + lectura[bit]) == 0)
                {
                    FX = false;
                }
                byteActual = byteActual + 1;
            }
        }

        //FRN=4, Data Item I020/140: Time of Day
        if (FSPEC[3])
        {
            lectura = BytesEnString(buffer, byteActual, 3);
            //Conversión de hora UTC a minutos y decimas de minuto
            //El LSB vale 1/128 segundos. Dividiendo el valor entre 128
obtenemos segundos
            int lecturaUTC = Convert.ToInt32(lectura, 2);       //Conversion a
decimal entero
            this.UTC=Convert.ToDouble(lecturaUTC) / (60 * 128);  //Al dividir se
obtienen decimales, pero eso conversion a double

            byteActual = byteActual + 3;
        }

        //FRN=5, Data Item I010/041: Position in WGS-84 Coordinates
        if (FSPEC[4])
        {
            string item41Lat = BytesEnString(buffer, byteActual, 4);
            byteActual = byteActual + 4;
            //obtencion del numero decimal de las coordenadas en grados
            double item41Latint =
Convert.ToDouble((ConvertirCA2aEntero(item41Lat)) * 180) / (Math.Pow(2,
31));
            string item41Long = BytesEnString(buffer, byteActual, 4);
            byteActual = byteActual + 4;
            double item41Longint =
Convert.ToDouble((ConvertirCA2aEntero(item41Long)) * 180) / (Math.Pow(2,
31));
            string[] WGS84Sign = new string[2];

            //Se rellena la matriz de signo
            if (item41Latint > 0)
            {
                WGS84Sign[0] = "N";
            }
            else
```

```
            {
                WGS84Sign[0] = "S";
            }
            if (item41Longint > 0)
            {
                WGS84Sign[1] = "E";
            }
            else
            {
                WGS84Sign[1] = "W";
            }
            this.WGS84Sign = WGS84Sign;

            //falta convertir los números decimales en 6 dígitos sin coma y sin
signo.
            //primero se elimina el simbolo y se combierte a string
            double item41LatintPos = Math.Abs(item41Latint);
            double item41LongintPos = Math.Abs(item41Longint);
            //convertir a string y en caso que se trate de un número inferior a 10
grados añadir un 0 al convertir a string para que no se confunda después. Ej:
2,2157º
            //en long, en caso de que sea inferior a 10º se añaden 2 ceros, si
menor a 100 se añade 1

            //2,2157891≠022157891 sino que es igual a :    2º ;
0.2157891*60=12.947346=12';  0.947346*60=56.84076=56"
            //0.84076*1000=840.76=841
            string LAT;
            string LONG;

            //Lat
            int grados = Convert.ToInt32(Math.Floor(item41LatintPos));
            int minutos;
            int segundos;
            int decimas;//3 decimas
            double decimales = (item41LatintPos - Convert.ToDouble(grados)) *
60;
            minutos = Convert.ToInt32(Math.Floor(decimales));
            decimales = (decimales - Convert.ToDouble(minutos)) * 60;
            segundos = Convert.ToInt32(Math.Floor(decimales));
            decimales = (decimales - Convert.ToDouble(segundos)) * 1000;
            decimas = Convert.ToInt32(Math.Round(decimales));
            string Grados = Convert.ToString(grados);
            string Minutos = Convert.ToString(minutos);
            string Segundos = Convert.ToString(segundos);
            string Decimas = Convert.ToString(decimas);

            if (grados < 10)
            {
                Grados = "0" + Grados;
```

```
}
if (minutos < 10)
{
    Minutos = "0" + Minutos;
}
if (segundos < 10)
{
    Segundos = "0" + Segundos;
}
if (decimas < 100)
{
    Decimas = "0" + Decimas;
    if (decimas < 10)
    {
        Decimas = "0" + Decimas;
    }
}
LAT = Grados + Minutos + Segundos + Decimas;

//Long
grados = Convert.ToInt32(Math.Floor(item41LongintPos));
decimales = (item41LongintPos - Convert.ToDouble(grados)) * 60;
minutos = Convert.ToInt32(Math.Floor(decimales));
decimales = (decimales - Convert.ToDouble(minutos)) * 60;
segundos = Convert.ToInt32(Math.Floor(decimales));
decimales = (decimales - Convert.ToDouble(segundos)) * 1000;
decimas = Convert.ToInt32(Math.Round(decimales));
Grados = Convert.ToString(grados);
Minutos = Convert.ToString(minutos);
Segundos = Convert.ToString(segundos);
Decimas = Convert.ToString(decimas);

if (grados < 100)
{
    Grados = "0" + Grados;
    if (grados < 10)
    {
        Grados = "0" + Grados;
    }
}
if (minutos < 10)
{
    Minutos = "0" + Minutos;
}
if (segundos < 10)
{
    Segundos = "0" + Segundos;
}
if (decimas < 100)
{
```

```
                Decimas = "0" + Decimas;
                if (decimas < 10)
                {
                    Decimas = "0" + Decimas;
                }
            }
            LONG = Grados + Minutos + Segundos + Decimas;
            this.WGS84Lat = LAT;//Latitud en WGS84. Sin signo, hay arriba una
variable para el signo
            this.WGS84Long = LONG;
            //string[] WGS84Sign; //Vector con 2 posiciones. En la primera "N" ó
"S"; en la segunda "E" ó "W". Ej: WGS84Sign=[N,E]
        }

        //FRN=6, Data Item I010/040: Measured Position in Polar Co-ordinates
        if (FSPEC[5])
        {
            //Rho
            lectura = BytesEnString(buffer, byteActual, 2);
            this.polarRho=Convert.ToInt32(lectura, 2); //LSB=1 m
            byteActual = byteActual + 2;
            //Theta
            lectura = BytesEnString(buffer, byteActual, 2);
            this.polarTheta=Convert.ToInt32(lectura, 2) * 0.0055;  //LSB=0.0055°
            byteActual = byteActual + 2;
        }

        //FRN=7, Data Item I010/042: Position in Cartesian Coordinates
(respecto al centro de Coordenadas)
        if (FSPEC[6])
        {
            //Coordenada X
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartX=ConvertirCA2aEntero(lectura);    //LSB = 1 m
            byteActual = byteActual + 2;
            //Coordenada Y
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartY=ConvertirCA2aEntero(lectura);    //LSB = 1 m
            byteActual = byteActual + 2;
        }

        //FRN=8, Data Item I010/200, Calculated Track Velocity in Polar Co-
ordinates
        if (FSPEC[7])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            this.polarTrackVelV=Convert.ToDouble(lectura) * 0.22;  //LSB=0.22 kt
            byteActual = byteActual + 2;
            lectura = BytesEnString(buffer, byteActual, 2);
```

```
            this.polarTrackVelAngle=Convert.ToDouble(lectura) * 0.0055;
//LSB=0.0055°
            byteActual = byteActual + 2;
        }

        //FRN=9, Data Item I010/202: Calculated Track Velocity in Cartesian
Coord.
        if (FSPEC[8])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartTrackVelX=ConvertirCA2aEntero(lectura) * 0.25;  //LSB=0.25
m/s
            byteActual = byteActual + 2;
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartTrackVelY=ConvertirCA2aEntero(lectura) * 0.25;
            byteActual = byteActual + 2;
        }

        //FRN=10, Data Item I010/161: Track Number
        if (FSPEC[9])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            lectura = Convert.ToString(Convert.ToInt32(lectura, 2));
            while (lectura.Length < 4)
                lectura = "0" + lectura;
            this.trackNumber= lectura;
            byteActual = byteActual + 2;
        }

        //FRN=11, Data Item I010/170: Track Status (Variable Length Data)
        if (FSPEC[10])
        {
            //Lectura de los bytes que forman el Data Item (mismo algoritmo que
Target Report Descriptor)
            FX = true;
            while (FX)
            {
                lectura = BytesEnString(buffer, byteActual, 1);
                bit = 0;
                while (bit < 7)
                {
                    bit = bit + 1;
                }
                if (Convert.ToInt32("" + lectura[bit]) == 0)
                {
                    FX = false;
                }
                byteActual = byteActual + 1;
            }
```

```
            //Some sensors are not be able to scan the whole coverage in one
refresh period,
            //therefore, track extrapolation is performed in un-scanned sectors.
CST is then set to 01.
        }

        //FRN=12, Data Item I010/060: Mode-3/A Code in Octal Representation
        if (FSPEC[11])
        {
            byteActual = byteActual + 2;
        }

        //FRN=13, Data Item I020/220: Target Address
        if (FSPEC[12])
        {
            lectura = BytesEnString(buffer, byteActual, 3);
            int ICAOAdd = Convert.ToInt32(lectura, 2);
            this.ICAOAdress=Convert.ToString(ICAOAdd,
16).ToUpper();//ToUpper convierte letras minusculas en mayusculas
            byteActual = byteActual + 3;
        }

        //FRN=14, Data Item I020/245: Target Identification (CallSign)
        if (FSPEC[13])
        {
            byteActual = byteActual + 1;
            //Los seis siguientes bits siempre son 0. Call sign comienza en el
segundo byte
            lectura = BytesEnString(buffer, byteActual, 6);
            //CallSign es formado por 8 caracteres de 6 bits cada uno
            //El algoritmo de decodificacion puede encontrarse en el Documento
ICAO Anexo 10, Volumen IV, sección 3.1.2.9
            this.callSign=TraducirCallSign(lectura);
            byteActual = byteActual + 6;
        }

        //FRN=15, Data Item I010/250: Mode S MB Data (Repetitive Data Item)
        if (FSPEC[14])
        {
            //No Decodificado, no llegan datos en el .ast CAT010
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt32(lectura, 2);
            byteActual = byteActual + 1 + numBytes;
        }

        //FRN=16, Data Item I010/300: Vehicle Fleet Identification
        if (FSPEC[15])
        {
            //No Decodificado, no llegan datos en el .ast CAT010
            byteActual = byteActual + 1;
```

```
        }

        //FRN=17, Data Item I020/090: Flight Level in Binary Representation
(Mode S Altitude)
        if (FSPEC[16])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            //Los dos primeros bits se almacenan en la variable "bitsVGdeFL"
            bit = 2;
            //Leemos el resto del Data Item
            string resultado = "";
            while (bit < 16)
            {
                resultado = resultado + lectura[bit];
                bit = bit + 1;
            }
            //Conversion de CA2 a decimal
            double result = ConvertirCA2aEntero(resultado) * 0.25;
            //Guardamos si el numero es negativo
            bool negativo = false;
            if (result < 0)
                negativo = true;
            result = Math.Abs(result);

            resultado = Convert.ToString(Math.Round(result));
            while (resultado.Length < 4)
                resultado = "0" + resultado;
            if (negativo)
                resultado = "-" + resultado;
            this.FL=Convert.ToString(resultado);

            byteActual = byteActual + 2;
        }

        //FRN=18, Data Item I020/110: Measured Height (Cartesian (Local)
Coordinates)
        if (FSPEC[17])
        {
            //No Decodificado, no llegan datos en el .ast CAT010
            byteActual = byteActual + 2;
        }

        //FRN=19, Data Item I010/270, Target Size & Orientation
        if (FSPEC[18])
        {
            lectura = BytesEnString(buffer, byteActual, 1);
            bit = 7;
            byteActual = byteActual + 1;
            if (Convert.ToInt32("" + lectura[bit]) == 1)
            {
```

```
            lectura = BytesEnString(buffer, byteActual, 1);
            bit = 7;
            byteActual = byteActual + 1;
            if (Convert.ToInt32("" + lectura[bit]) == 1)
            {
                lectura = BytesEnString(buffer, byteActual, 1);
                bit = 7;
                byteActual = byteActual + 1;
            }
        }
    }

    //FRN=20, Data Item I010/550, System Status
    if (FSPEC[19])
    {
        //Decodificación no necesaria
        byteActual = byteActual + 1;
    }

    //FRN=21, Data Item I020/310: Pre-programmed Message
    if (FSPEC[20])
    {
        //No Decodificado, no llegan datos en el .ast CAT010
        byteActual = byteActual + 1;
    }

    //FRN=22, Data Item I010/500, Standard Deviation of Position
    if (FSPEC[21])
    {
        //No Decodificado, no llegan datos en el .ast CAT010
        byteActual = byteActual + 4;
    }

    //FRN=23, Data Item I010/280, Presence
    if (FSPEC[22])
    {
        ///No Decodificado, no llegan datos en el .ast CAT010
        lectura = BytesEnString(buffer, byteActual, 1);
        int numBytes = Convert.ToInt32(lectura, 2);
        byteActual = byteActual + 1 + numBytes;
    }

    //FRN=24, Data Item I010/131, Amplitude of Primary Plot
    if (FSPEC[23])
    {
        //No Decodificado, no llegan datos en el .ast CAT010
        byteActual = byteActual + 1;
    }

    //FRN=25, Data Item I010/210: Calculated Acceleration
```

```
        if (FSPEC[24])
        {
            lectura = BytesEnString(buffer, byteActual, 1);
            this.accX = ConvertirCA2aEntero(lectura) * 0.25;  //LSB=0.25 m/s^2
            byteActual = byteActual + 1;
            lectura = BytesEnString(buffer, byteActual, 1);
            this.accY = ConvertirCA2aEntero(lectura) * 0.25;
            byteActual = byteActual + 1;
        }

        //FRN=27, Data Item SP: Special Purpose Field
        if (FSPEC[26])
        {
            //No Decodificado, no llegan datos en el .ast CAT010
            //El primer byte indica la Length de SP, incluido este primer byte
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt16(lectura);
            byteActual = byteActual + numBytes;
        }

        //FRN=28, Data Item RE: Reserved Expansion Field
        if (FSPEC[27])
        {
            //No Decodificado, no llegan datos en el .ast CAT010
            //El primer byte indica la Length de RE, incluido este primer byte
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt16(lectura);
            byteActual = byteActual + numBytes;
        }
    }

    public void DecodificarCAT20(FileStream fs, byte[] buffer, int byteActual)
    {
        //Leemos la Length del mensaje
        fs.Read(buffer, byteActual, 2);
        int LEN = Convert.ToInt32(BytesEnString(buffer, 1, 2), 2);
        byteActual = byteActual + 2;

        //Guardamos el resto del mensaje en buffer
        fs.Read(buffer, byteActual, LEN - 3);

        //Leemos FSPEC
        string lectura;          //Guarda el byte actual de FSPEC en un string
        int bit;                 //Contador del bit actual que se está leyendo
        bool[] FSPEC = new bool[28];    //el vector indica qué Data Items se han
enviado en el mensaje
        int FRN = 0;                //Indica la posicion actual de FSPEC

        bool FX = true; //Cuando se lee FSPEC, si el último bit es 1 se sigue
leyendo el siguiente byte. Si no, acaba FSPEC.
```

```
        while (FX)
        {
            lectura = BytesEnString(buffer, byteActual, 1);

            //Leemos los bits y guardamos qué Data Items se han enviado
            bit = 0;
            while (bit < 7)
            {
                if (Convert.ToInt32("" + lectura[bit]) == 1)
                    FSPEC[FRN] = true;
                if (Convert.ToInt32("" + lectura[bit]) == 0)
                    FSPEC[FRN] = false;

                bit = bit + 1;
                FRN = FRN + 1;
            }
            //leemos último bit
            if (Convert.ToInt32("" + lectura[bit]) == 0)
                FX = false;

            byteActual = byteActual + 1;
        }

        //Lectura de cada uno de los Data Fields

        //FRN=1, Data Item I020/010: Data Source Identifier
        if (FSPEC[0])
        {
            //SAC
            lectura = Convert.ToString(buffer[byteActual], 10);
            while (lectura.Length < 3)
                lectura = "0" + lectura;
            this.SAC = lectura;
            byteActual++;

            //SIC
            lectura = Convert.ToString(buffer[byteActual], 10);
            while (lectura.Length < 3)
                lectura = "0" + lectura;
            this.SIC = lectura;
            byteActual++;
        }

        //FRN=2, Data Item I020/020: Target Report Descriptor (Variable Length
Data)
        if (FSPEC[1])
        {
            //Lectura de los bytes que forman el Data Item (mismo algoritmo que
FSPEC)
            FX = true;
```

```
            while (FX)
            {
                lectura = BytesEnString(buffer, byteActual, 1);
                bit = 0;
                while (bit < 7)
                {
                    bit = bit + 1;
                }
                if (Convert.ToInt32("" + lectura[bit]) == 0)
                {
                    FX = false;
                }
                byteActual = byteActual + 1;
            }
        }

        //FRN=3, Data Item I020/140: Time of Day
        if (FSPEC[2])
        {
            lectura = BytesEnString(buffer, byteActual, 3);
            //Conversión de hora UTC a minutos y decimas de minuto
            //El LSB vale 1/128 segundos. Dividiendo el valor entre 128
obtenemos segundos
            int lecturaUTC = Convert.ToInt32(lectura, 2);       //Conversion a
decimal entero
            this.UTC = Convert.ToDouble(lecturaUTC) / (60 * 128);  //Al dividir se
obtienen decimales, pero eso conversion a double

            byteActual = byteActual + 3;
        }

        //FRN=4, Data Item I020/041: Position in WGS-84 Coordinates
        if (FSPEC[3])
        {
            //Decodificación no requerida
            byteActual = byteActual + 8;
        }

        //FRN=5, Data Item I020/042: Position in Cartesian Coordinates
(respecto al centro de Coordenadas)
        if (FSPEC[4])
        {
            //Coordenada X
            lectura = BytesEnString(buffer, byteActual, 3);
            this.cartX = ConvertirCA2aEntero(lectura) * 0.5;    //LSB = 0.5 m
            byteActual = byteActual + 3;
            //Coordenada Y
            lectura = BytesEnString(buffer, byteActual, 3);
            this.cartY = ConvertirCA2aEntero(lectura) * 0.5;    //LSB = 0.5 m
            byteActual = byteActual + 3;
```

```
        }

        //FRN=6, Data Item I020/161: Track Number
        if (FSPEC[5])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            lectura = Convert.ToString(Convert.ToInt32(lectura, 2));
            while (lectura.Length < 4)
                lectura = "0" + lectura;
            this.trackNumber = lectura;
            byteActual = byteActual + 2;
        }

        //FRN=7, Data Item I020/170: Track Status (Variable Length Data)
        if (FSPEC[6])
        {
            //Lectura de los bytes que forman el Data Item (mismo algoritmo que
Target Report Descriptor)
            FX = true;
            while (FX)
            {
                lectura = BytesEnString(buffer, byteActual, 1);
                bit = 7;
                if (Convert.ToInt32("" + lectura[bit]) == 0)
                {
                    FX = false;
                }
                byteActual = byteActual + 1;
            }
        }

        //FRN=8, Data Item I020/070: Mode-3/A Code in Octal Representation
        if (FSPEC[7])
        {
            byteActual = byteActual + 2;
        }

        //FRN=9, Data Item I020/202: Calculated Track Velocity in Cartesian
Coord.
        if (FSPEC[8])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartTrackVelX = ConvertirCA2aEntero(lectura) * 0.25;
            byteActual = byteActual + 2;
            lectura = BytesEnString(buffer, byteActual, 2);
            this.cartTrackVelY = ConvertirCA2aEntero(lectura) * 0.25;
            byteActual = byteActual + 2;
        }
```

```
        //FRN=10, Data Item I020/090: Flight Level in Binary Representation
(Mode S Altitude)
        if (FSPEC[9])
        {
            lectura = BytesEnString(buffer, byteActual, 2);
            //Los dos primeros bits se almacenan en la variable "bitsVGdeFL"
            bit = 2;
            //Leemos el resto del Data Item
            string resultado = "";
            while (bit < 16)
            {
                resultado = resultado + lectura[bit];
                bit = bit + 1;
            }
            //Conversion de CA2 a decimal
            double result = ConvertirCA2aEntero(resultado) * 0.25;
            //Guardamos si el numero es negativo
            bool negativo = false;
            if (result < 0)
                negativo = true;
            result = Math.Abs(result);

            resultado = Convert.ToString(Math.Round(result));
            while (resultado.Length < 4)
                resultado = "0" + resultado;
            if (negativo)
                resultado = "-" + resultado;
            this.FL = Convert.ToString(resultado);

            byteActual = byteActual + 2;
        }

        //FRN=11, Data Item I020/100: Mode-C Code
        if (FSPEC[10])
        {
            //Decodificación no requerida
            byteActual = byteActual + 4;
        }

        //FRN=12, Data Item I020/220: Target Address
        if (FSPEC[11])
        {
            lectura = BytesEnString(buffer, byteActual, 3);
            int ICAOAdd = Convert.ToInt32(lectura, 2);
            this.ICAOAdress = Convert.ToString(ICAOAdd, 16).ToUpper();
            byteActual = byteActual + 3;
        }

        //FRN=13, Data Item I020/245: Target Identification (CallSign)
        if (FSPEC[12])
```

```
        {
            byteActual = byteActual + 1;
            //Los seis siguientes bits siempre son 0. Call sign comienza en el
segundo byte
            lectura = BytesEnString(buffer, byteActual, 6);
            //CallSign es formado por 8 caracteres de 6 bits cada uno
            //El algoritmo de decodificacion puede encontrarse en el Documento
ICAO Anexo 10, Volumen IV, sección 3.1.2.9
            this.callSign = TraducirCallSign(lectura);
            byteActual = byteActual + 6;
        }

        //FRN=14, Data Item I020/110: Measured Height (Cartesian (Local)
Coordinates)
        if (FSPEC[13])
        {
            //Decodificación no requerida
            byteActual = byteActual + 2;
        }

        //FRN=15, Data Item I020/105: Geometric Height (WGS-84)
        if (FSPEC[14])
        {
            //Decodificación no requerida
            byteActual = byteActual + 2;
        }

        //FRN=16, Data Item I020/210: Calculated Acceleration
        if (FSPEC[15])
        {
            //Decodificación no requerida
            byteActual = byteActual + 2;
        }

        //FRN=17, Data Item I020/300: Vehicle Fleet Identification
        if (FSPEC[16])
        {
            //Decodificación no requerida
            byteActual = byteActual + 1;
        }

        //FRN=18, Data Item I020/310: Pre-programmed Message
        if (FSPEC[17])
        {
            //Decodificación no requerida
            byteActual = byteActual + 1;
        }

        //FRN=19, Data Item I020/500: Position Accuracy (DOP) (Compound
Data Item)
```

```
        if (FSPEC[18])
        {
            //Primary Subfield: el primer byte indica el contenido de los siguientes
bytes
            //(En principio sólo los tres primeros bits indican el contenido, los
otros 5 deberían ser "spare bits set to zero")
            bool[] primarySub = new bool[8];

            lectura = BytesEnString(buffer, byteActual, 1);
            bit = 0;
            while (bit < 8)
            {
                if (Convert.ToInt16("" + lectura[bit]) == 1)
                    primarySub[bit] = true;
                if (Convert.ToInt16("" + lectura[bit]) == 0)
                    primarySub[bit] = false;
                bit++;
            }

            byteActual++;

            //Subfield #1 (bit DOP true)
            if (primarySub[0])
            {
                byteActual = byteActual + 2;
            }

            //Subfield #2 (bit SDP true)
            if (primarySub[1])
            {
                byteActual = byteActual + 2;
            }

            //Subfield #3 (bit SDH true)
            if (primarySub[2])
            {
                byteActual = byteActual + 2;
            }
        }

        //FRN=20, Data Item I020/400: Contributing Devices (Repetitive Data
Item)
        if (FSPEC[19])
        {
            //El primer byte indica el numero de bytes a leer después de este
            lectura = BytesEnString(buffer, byteActual, 1);
            byteActual++;
            int numBytes = Convert.ToInt32(lectura, 2);
            if (numBytes != 0)
            {
```

```
                byteActual = byteActual + numBytes;
            }
        }

        //FRN=21, Data Item I020/250: Mode S MB Data (Repetitive Data Item)
        if (FSPEC[20])
        {
            //Decodificación no requerida
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt32(lectura, 2);
            byteActual = byteActual + 1 + numBytes;
        }

        //FRN=22, Data Item I020/230: Comms/ACAS Capability and Flight
Status (Two-octet fixed data length)
        if (FSPEC[21])
        {
            //Decodificación no requerida
            byteActual = byteActual + 2;
        }

        //FRN=23, Data Item I020/260: ACAS Resolution Advisory Report
(Seven-octet fixed data length)
        if (FSPEC[22])
        {
            //Decodificación no requerida
            byteActual = byteActual + 7;
        }

        //FRN=24, Data Item I020/030: Warning/Error Conditions (Variable
length Data Item)
        if (FSPEC[23])
        {
            //Decodificación no requerida
            FX = true;
            while (FX)
            {
                lectura = BytesEnString(buffer, byteActual, 1);
                if (Convert.ToInt16("" + lectura[7]) == 0)
                    FX = false;
                byteActual++;
            }
        }

        //FRN=25, Data Item I020/055: Mode-1 Code in Octal Representation (1
byte)
        if (FSPEC[24])
        {
            //Decodificación no requerida
            byteActual++;
```

```
        }

        //FRN=26, Data Item I020/050: Mode-2 Code in Octal Representation (2
bytes)
        if (FSPEC[25])
        {
            //Decodificación no requerida
            byteActual = byteActual + 2;
        }

        //FRN=27, Data Item RE: Reserved Expansion Field
        if (FSPEC[26])
        {
            //El primer byte indica la Length de RE, incluido este primer byte
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt16(lectura) - 1;//restamos 1 para no
contar el de LEN
            byteActual = byteActual + 1;

            if (numBytes > 0)
            {
                bool[] subFSPEC = new bool[8];
                lectura = BytesEnString(buffer, byteActual, 1);
                bit = 0;
                int indice = 0;
                while (bit < 8)
                {
                    if (Convert.ToInt32("" + lectura[bit]) == 1)
                        subFSPEC[indice] = true;
                    if (Convert.ToInt32("" + lectura[bit]) == 0)
                        subFSPEC[indice] = false;

                    bit = bit + 1;
                    indice = indice + 1;
                }
                byteActual = byteActual + 1;

                //PA, Position Accuracy
                //Si llega, la tabla PA substituye la tabla que se lee en el item PA
                if (subFSPEC[0])
                {
                    bool[] primarySub = new bool[8];
                    lectura = BytesEnString(buffer, byteActual, 1);
                    bit = 0;
                    while (bit < 8)
                    {
                        if (Convert.ToInt16("" + lectura[bit]) == 1)
                            primarySub[bit] = true;
                        if (Convert.ToInt16("" + lectura[bit]) == 0)
                            primarySub[bit] = false;
```

```
                bit++;
            }
            byteActual = byteActual + 1;

            //Subfield #1: DOP of Position
            if (primarySub[0])
            {
                //DOPx
                byteActual = byteActual + 2;
                //DOPy
                byteActual = byteActual + 2;
                //DOPxy
                //Si DOPx o DOPy valen 0, entonces DOPxy vale 0
                byteActual = byteActual + 2;
            }

            //Subfield #2 Standard Deviation of Position (Cartesian)
            if (primarySub[1])
            {
                //Standard Deviation of X component
                byteActual = byteActual + 2;
                //Standard Deviation of Y component
                byteActual = byteActual + 2;
                //Coeficiente de Correlación, en Complemento a 2
                //Si stDevX o stDevY valen 0, entonces el coef. de correlacion
vale 0
                byteActual = byteActual + 2;
            }

            //Subfield #3 Standard Deviation of Geometric Height
            if (primarySub[2])
            {
                //Standard Deviation of Height
                byteActual = byteActual + 2;
            }

            //Subfield #4 Standard Deviation of Position (WGS-84)
            if (primarySub[3])
            {
                //No decodificamos
                byteActual = byteActual + 6;
            }
            //El resto son spare
        }
        if (subFSPEC[1])
        {
            //No decodificamos
            byteActual = byteActual + 4;
        }
        if (subFSPEC[2])
```

```
{
    //No decodificamos
    byteActual = byteActual + 2;
}
if (subFSPEC[3])
{
    //No decodificamos
    byteActual = byteActual + 3;
}
if (subFSPEC[4])
{
    //No decodificamos
    lectura = BytesEnString(buffer, byteActual, 1);
    bool[] primerSub = new bool[8];
    bool[] segundoSub = new bool[8];
    bool[] tercerSub = new bool[3];
    bit = 0;
    while (bit < 8)
    {
        if (Convert.ToInt16("" + lectura[bit]) == 1)
            primerSub[bit] = true;
        if (Convert.ToInt16("" + lectura[bit]) == 0)
            primerSub[bit] = false;
        bit++;
    }
    byteActual = byteActual + 1;

    if (primerSub[7])
    {
        lectura = BytesEnString(buffer, byteActual, 1);
        bit = 0;
        while (bit < 8)
        {
            if (Convert.ToInt16("" + lectura[bit]) == 1)
                segundoSub[bit] = true;
            if (Convert.ToInt16("" + lectura[bit]) == 0)
                segundoSub[bit] = false;
            bit++;
        }
        byteActual = byteActual + 1;

        if (segundoSub[7])
        {
            lectura = BytesEnString(buffer, byteActual, 1);
            bit = 0;
            while (bit < 3)
            {
                if (Convert.ToInt16("" + lectura[bit]) == 1)
                    tercerSub[bit] = true;
                if (Convert.ToInt16("" + lectura[bit]) == 0)
```

```
                tercerSub[bit] = false;
            bit++;
        }
        byteActual = byteActual + 1;
    }
}
if (primerSub[0])
{
    byteActual = byteActual + 1;
}
if (primerSub[1])
{
    byteActual = byteActual + 1;
}
if (primerSub[2])
{
    byteActual = byteActual + 3;
}
if (primerSub[3])
{
    byteActual = byteActual + 1;
}
if (primerSub[4])
{
    byteActual = byteActual + 1;
}
if (primerSub[5])
{
    byteActual = byteActual + 1;
}
if (primerSub[6])
{
    byteActual = byteActual + 1;
}
if (segundoSub[0])
{
    byteActual = byteActual + 1;
}
if (segundoSub[1])
{
    byteActual = byteActual + 1;
}
if (segundoSub[2])
{
    byteActual = byteActual + 3;
}
if (segundoSub[3])
{
    byteActual = byteActual + 1;
}
```

```
                    if (segundoSub[4])
                    {
                        byteActual = byteActual + 1;
                    }
                    if (segundoSub[5])
                    {
                        byteActual = byteActual + 1;
                    }
                    if (segundoSub[6])
                    {
                        byteActual = byteActual + 1;
                    }
                    if (tercerSub[0])
                    {
                        byteActual = byteActual + 1;
                    }
                    if (tercerSub[1])
                    {
                        byteActual = byteActual + 1;
                    }
                    if (tercerSub[2])
                    {
                        byteActual = byteActual + 3;
                    }
                }
                //El resto son spare
            }
        }

        //FRN=28, Data Item SP: Special Purpose Field
        if (FSPEC[27])
        {
            //Decodificación no requerida
            //El primer byte indica la Length de SP, incluido este primer byte
            lectura = BytesEnString(buffer, byteActual, 1);
            int numBytes = Convert.ToInt16(lectura);
            byteActual = byteActual + numBytes;
        }
    }

    public void DecodificarDGPS(string linea, string Icao) //Lee el .txt de D-GPS y le asigna la misma estructura que al fichero .ast
    {
        this.CAT = 10;
        this.SIC = "107";  //Predeterminado: origen MLAT, ICAO Address introducido por usuario para todos los paquetes, datos centrados (referenciados) en MLAT.
        this.SAC = "000";
        this.ICAOAdress = Icao;
```

```
            string[] entradas = linea.Split('\t');
            string[] entradasTime = entradas[5].Split(' ');
            if (entradasTime[0] == "")
            {
                if (entradasTime[3] == "")
                {
                    entradasTime[0] = entradasTime[1];
                    entradasTime[1] = entradasTime[2];
                    entradasTime[2] = entradasTime[4];
                }
                else
                {
                    entradasTime[0] = entradasTime[1];
                    entradasTime[1] = entradasTime[2];
                    entradasTime[2] = entradasTime[3];
                }
            }
            else if (entradasTime[2] == "")
            {
                entradasTime[2] = entradasTime[3];
            }
            double a = Convert.ToDouble(entradasTime[2]);
            this.UTC = (Convert.ToDouble(entradasTime[0]) * 60) +
Convert.ToDouble(entradasTime[1]) + (Convert.ToDouble(entradasTime[2]) /
(60 * 100));//Pasamos a minutos y decimas de minuto
            string[] Mlat = new string[] { "411749426N", "0020442410E"
};//Coordenadas del mlat
            double[] cartMlat = new double[2];
            double[] posicion = new double[2];
            cartMlat[0] = ConvertWGStoRad(Mlat[0], 0);//Mlat en radianes
            cartMlat[1] = ConvertWGStoRad(Mlat[1], 1);
            posicion = ConvertRadtoXY(cartMlat, Convert.ToDouble(entradas[1]) *
Math.PI / (180 * Math.Pow(10, 10)), Convert.ToDouble(entradas[2]) * Math.PI /
(180 * Math.Pow(10, 10)));// /Math.Pow(10, 10) porque no transforma bien el .
decimal
            this.cartX = posicion[0];//Pasamos WGS a XY
            this.cartY = posicion[1];
        }

        //Subfunciones
        public double[] ConvertRadtoXY(double[] llh0, double dphi, double
dlam)//proyecta WGS84 sobre el plano. Lat long en radianes
        {
            double a = 6378137;
            double b = 6356752.3142;
            double e2;
            e2 = 1 - Math.Pow((b / a), 2);
            //Location of reference point in radians
            double phi = llh0[0];//lat
            double lam = llh0[1];//long
```

```
                    //Some useful definitions
            double tmp1 = Math.Sqrt(1 - e2 * Math.Pow(Math.Sin(phi), 2));
            double cl = Math.Cos(lam);
            double sl = Math.Sin(lam);
            double cp = Math.Cos(phi);
            double sp = Math.Sin(phi);
            //Location of data points in radians
            double[] denu = new double[2];
            dphi = dphi - phi;
            dlam = dlam - lam;
            //Transformations
            denu[0] = (a / tmp1) * cp * dlam - (a * (1 - e2) / (Math.Pow(tmp1, 3))) * sp
* dphi * dlam; //X
            denu[1] = (a * (1 - e2) / Math.Pow(tmp1, 3)) * dphi + 1.5 * cp * sp * a * e2
* Math.Pow(dphi, 2) + 0.5 * sp * cp * (a / tmp1) * Math.Pow(dlam, 2); //Y

            return (denu);
        }

        public double ConvertWGStoRad(string coordenada, int orientacion)//Si es
lat, orientacion=0; Si es long, orientacion=1
        {
            //Retorna una coordenada en radianes //El imput es sin decimas de
segundo
            double grados = 0;
            int latSigno = 1; //por defecto es positivo para Norte
            int longSigno = 1; //por defecto es positivo para Este
            if (orientacion == 0)
            {
                if (Convert.ToString(coordenada[9]) == "S")
                {
                    latSigno = -1;
                }

                grados = Convert.ToDouble(Convert.ToString(coordenada[0]) +
Convert.ToString(coordenada[1]));
                double minutos =
Convert.ToDouble(Convert.ToString(coordenada[2]) +
Convert.ToString(coordenada[3]));
                double segundos =
Convert.ToDouble(Convert.ToString(coordenada[4]) +
Convert.ToString(coordenada[5]));
                double decimas =
Convert.ToDouble(Convert.ToString(coordenada[6]) +
Convert.ToString(coordenada[7]) + Convert.ToString(coordenada[8])) / 1000;
                segundos = segundos + decimas;
                grados = grados + (minutos / 60) + (segundos / 3600);
                grados = latSigno * grados * Math.PI / 180;
            }
            if (orientacion == 1)
```

```
        {
            if (Convert.ToString(coordenada[10]) == "W")
            {
                longSigno = -1;
            }

            grados = Convert.ToDouble(Convert.ToString(coordenada[0]) +
Convert.ToString(coordenada[1]) + Convert.ToString(coordenada[2]));
            double minutos =
Convert.ToDouble(Convert.ToString(coordenada[3]) +
Convert.ToString(coordenada[4]));
            double segundos =
Convert.ToDouble(Convert.ToString(coordenada[5]) +
Convert.ToString(coordenada[6]));
            double decimas =
Convert.ToDouble(Convert.ToString(coordenada[7]) +
Convert.ToString(coordenada[8]) + Convert.ToString(coordenada[9])) / 1000;
            segundos = segundos + decimas;

            grados = grados + (minutos / 60) + (segundos / 3600);
            grados = longSigno * grados * Math.PI / 180;
        }
        return (grados);
    }

    public void ConvertCartFromPolar() //int[x,y] //No se usa
    {
        //CAT10 usa SMR para datos polares, por lo que esta centrado en
SMR. Le sumamos //SMR y le restamos MLAT para referenciarlo a MLAT.
Theta esta referencia en el norte
        double Rt = 6378000;//m Radio ecuatorial
        string[] Mlat = new string[] { "411749426N", "0020442410E" };
//Coordenadas del MLAT y SMR
        string[] SMR = new string[] { "411744226N", "0020542411E" };
        double[] cartMlat = new double[2];
        double[] cartSMR = new double[2];
        cartMlat[0] = Rt * ConvertWGStoRad(Mlat[1], 1);
        cartMlat[1] = Rt * ConvertWGStoRad(Mlat[0], 0);
        cartSMR[0] = Rt * ConvertWGStoRad(SMR[1], 1);
        cartSMR[1] = Rt * ConvertWGStoRad(SMR[0], 0);
        if ((this.polarRho != Math.Pow(10, 8)) && (this.polarTheta !=
Math.Pow(10, 8)))
        {
            this.cartFromPolarX = Convert.ToInt32(this.polarRho *
Math.Sin(this.polarTheta) + cartSMR[0] - cartMlat[0]);
            this.cartFromPolarY = Convert.ToInt32(this.polarRho *
Math.Cos(this.polarTheta) + cartSMR[1] - cartMlat[1]);
        }
    }
```

```
        public string ConvertUTC(string decimas) //Pasa UTC de minutos y
decimas de minuto a un string en HH:mm::ss.sss
        {
            //Si decimas="SinDecimas" retorna formato hh:mm:ss; Sino dejar
decimas="".
            double minutes = this.UTC;
            string resultado;
            if (minutes != -1)
            {
                double hours = Math.Floor(minutes / 60);
                double roundMinutes = Math.Floor(minutes - hours * 60);
                double seconds;
                if (decimas == "")
                    seconds = Math.Round((minutes - (hours * 60) - roundMinutes) *
60, 3);//Permitimos 3 decimas de segundo
                else
                    seconds = Math.Round((minutes - (hours * 60) - roundMinutes) *
60);
                string finalHours = Convert.ToString(hours);
                string finalMinutes = Convert.ToString(roundMinutes);
                string finalSeconds=Convert.ToString(seconds);
                if (hours < 10)
                    finalHours = "0" + finalHours;
                if (roundMinutes < 10)
                    finalMinutes = "0" + finalMinutes;
                if (seconds < 10)
                    finalSeconds = "0" + finalSeconds;
                string prefinal = finalHours + ":" + finalMinutes + ":" + finalSeconds;
                if (decimas == "")
                {
                    if (prefinal.Length == 8)
                    {
                        resultado = prefinal + ",000";
                    }
                    else if (prefinal.Length == 10)
                    {
                        resultado = prefinal + "00";
                    }
                    else if (prefinal.Length == 11)
                    {
                        resultado = prefinal + "0";
                    }
                    else resultado = prefinal;
                }
                else resultado = prefinal;
            }
            else resultado = "No Data";

            return resultado;
        }
```

```
        public double[] GetPosicion() //Miramos si los datos de posicion del
paquete vienen en WGS, del SMR, o en coordenadas cartesianas del MLAT
        {
            double[] posicion = new double[2];//Indice 0=X; indice 1=Y
            if ((GetCartFromWGS84()[0] != Math.Pow(10, 8)) &&
(GetCartFromWGS84()[1] != Math.Pow(10, 8)))
            {
                posicion[0] = GetCartFromWGS84()[0];
                posicion[1] = GetCartFromWGS84()[1];
            }
            else if ((GetMLATfromSMRX() != Math.Pow(10, 8)) &&
(GetMLATfromSMRY() != Math.Pow(10, 8)))
            {
                posicion[0] = GetMLATfromSMRX();
                posicion[1] = GetMLATfromSMRY();
            }
            else
            {
                posicion[0] = GetCartX();
                posicion[1] = GetCartY();
            }
            return (posicion);
        }

        //Funciones GET

        public int GetCAT()
        {
            return (this.CAT);
        }
        public string GetSIC()
        {
            return (this.SIC);
        }
        public string GetSAC()
        {
            return (this.SAC);
        }
        public double GetUTC()
        {
            return (this.UTC);
        }
        public string GetTrackNumber()
        {
            return (this.trackNumber);
        }
        public string GetFL()
        {
            return (this.FL);
```

```
        }
        public string GetCallSign()
        {
            return (this.callSign);
        }
        public string GetICAOAdress()
        {
            return (this.ICAOAdress);
        }
        public double GetCartX()
        {
            return (this.cartX);
        }
        public double GetCartY()
        {
            return (this.cartY);
        }
        public double GetPolarRho()
        {
            return (this.polarRho);
        }
        public double GetPolarTheta()
        {
            return (this.polarTheta);
        }
        public double GetCartFromPolarX()
        {
            return (this.cartFromPolarX);
        }
        public double GetCartFromPolarY()
        {
            return (this.cartFromPolarY);
        }
        public string GetWGS84Lat()
        {
            //Retornamos todo menos las decimas. Lat 41 51 23 111
            if (this.WGS84Lat != null)
            {
                return (Convert.ToString(this.WGS84Lat[0]) +
Convert.ToString(this.WGS84Lat[1]) + Convert.ToString(this.WGS84Lat[2]) +
Convert.ToString(this.WGS84Lat[3]) +
                    Convert.ToString(this.WGS84Lat[4]) +
Convert.ToString(this.WGS84Lat[5]));
            }
            else return (this.WGS84Lat);
        }
        public string GetWGS84Long()
        {
            //Retornamos todo menos las decimas. Lat 041 51 23 111
            if (this.WGS84Long != null)
```

```
        {
            return (Convert.ToString(this.WGS84Long[0]) +
Convert.ToString(this.WGS84Long[1]) + Convert.ToString(this.WGS84Long[2])
+ Convert.ToString(this.WGS84Long[3]) +
                Convert.ToString(this.WGS84Long[4]) +
Convert.ToString(this.WGS84Long[5]) +
Convert.ToString(this.WGS84Long[6]));
        }
        else return (this.WGS84Long);
    }
    public string GetWGS84Sign(int componente)
    {
        return (this.WGS84Sign[componente]);
    }
    public double GetCartTrackVelX()
    {
        return (this.cartTrackVelX);
    }
    public double GetCartTrackVelY()
    {
        return (this.cartTrackVelY);
    }
    public double GetPolarTrackVelV()
    {
        return (this.polarTrackVelV);
    }
    public double GetPolarTrackVelAngle()
    {
        return (this.polarTrackVelAngle);
    }
    public double GetMLATfromSMRX()
    {
        return (this.MLATfromSMRX);
    }
    public double GetMLATfromSMRY()
    {
        return (this.MLATfromSMRY);
    }
    public double GetAccX()
    {
        return (this.accX);
    }
    public double GetAccY()
    {
        return (this.accY);
    }
    public int GetZona()
    {
        return (this.zona);
    }
```

```
        public int GetIndiceDGPS()
        {
            return (this.indiceDGPS);
        }
        public double GetUTCcorregido()
        {
            return (this.UTCcorregido);
        }


        //Funciones SET
        public void SetZona(int Zona)
        {
            this.zona = Zona;
        }
        public void SetIndiceDGPS(int indice)
        {
            this.indiceDGPS = indice;
        }
        public void SetUTCcorregido(double UTCc)
        {
            this.UTCcorregido = UTCc;
        }

        public void SetVelocidad(double Vx, double Vy)
        {
            this.cartTrackVelX = Vx; //Cartesian track velocity (X component).
            this.cartTrackVelY = Vy;
        }
    }
}
```

LectorMensaje.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;

namespace Codigo
{
   public class LectorMensaje
   {
      List<DecodificadorMensaje> myList = new List<DecodificadorMensaje>();

      public int GetNumList()
      {
         return myList.Count();
      }

      public DecodificadorMensaje GetPlanI(int i)
      {
         return myList[i];
      }

      public void RemovePlanI(int i)
      {
         myList.RemoveAt(i);
      }

      public void AddPlanI(DecodificadorMensaje plan)
      {
         myList.Add(plan);
      }

      public void ClearList()
      {
         myList.Clear();
      }

      public int AircraftDetected() //Retorna el recuento de aviones (compara
ICAO Adress). Retorna 0 si el item no llega
      {
         int indice = 0;
         List<string> listaAviones = new List<string>();

         while ((myList[indice].GetICAOAdress() == "No Data") && (indice <
myList.Count() - 1))
         {
            indice++;
         }
```

```
            if (indice < myList.Count() - 1)
            {
                listaAviones.Add(myList[indice].GetICAOAdress());
                for (int i = indice + 1; i < myList.Count(); i++)
                {
                    if (myList[i].GetICAOAdress() != "No Data")
                    {
                        bool yaExiste = false;
                        int n = 0;
                        while ((n < listaAviones.Count()) && (!yaExiste))
                        {
                            if (myList[i].GetICAOAdress() == listaAviones[n])
                            {
                                yaExiste = true;
                            }
                            else n++;
                        }
                        if (!yaExiste)
                            listaAviones.Add(myList[i].GetICAOAdress());
                    }
                }
                return (listaAviones.Count());
            }
            else return (0);
        }

        public void CargarListaDeDirectorio(string nombreDirectorio)
        //Rellena la lista de mensajes desde un fichero cualquiera
        {
            FileStream fs = new FileStream(nombreDirectorio, FileMode.Open,
FileAccess.Read);
            myList.Clear();
            bool yaEsta = false;
            DecodificadorMensaje mensaje;

            //Leemos mensajes hasta que encontremos un mensaje vacío
            int CatInicial = 0;
            while (!yaEsta)
            {
                mensaje = new DecodificadorMensaje();
                mensaje.LeerSiguienteMensaje(fs);
                //Si el mensaje está vacío, se descarta y se deja de leer
                if ((CatInicial != 10)&&(CatInicial != 20))
                    CatInicial = mensaje.GetCAT();//Solo podremos leer un tipo de
CAT a la vez (10 ó 20)
                if (mensaje.GetCAT() == -1)
                    yaEsta = true;
                else if ((mensaje.GetCAT() == 10) && (CatInicial == 10))
                {
                    myList.Add(mensaje);
```

```
            }
            else if ((mensaje.GetCAT() == 20) && (CatInicial == 20))
            {
                myList.Add(mensaje);
            }
        }
        fs.Close();
        SetZonas();//Asignamos la zona en funcion de la posicion en el mapa
        //Antes de OrdenarPaquetesPorTiempo, descartamos SMR. Antes de
asignarUTCcorregido ordenamos por tiempo.
        //Por lo tanto ambas funciones se ejecutan en Form1, despues de
descartar los vehiculos SMR.
    }

    public void CargarListaDeDirectorioDGPS(string nombreDirectorio, string
Icao)
    //Rellena la lista de mensajes desde un fichero cualquiera
    {
        StreamReader leer = new StreamReader(nombreDirectorio);
        myList.Clear();
        DecodificadorMensaje mensaje;

        //Leemos mensajes hasta que encontremos un mensaje vacío
        string linea = leer.ReadLine();

        bool datos = false;
        while ((linea != null) && (!datos))
        {
            string a = linea.Split('\t')[1];
            double numero = 0;
            if (!double.TryParse(a, out numero))//A veces la primera linea no
contiene datos sino info de la estructura de los datos, la saltamos
            {
                //TryParse() permite convertir y trabajar con la respuesta
                //si se puede realizar correctamente el resultado se asigna en la
variable definida en el out
                linea = leer.ReadLine();
            }
            else datos = true;
        }

        while (linea != null)
        {
            mensaje = new DecodificadorMensaje();
            mensaje.DecodificarDGPS(linea, Icao);
            myList.Add(mensaje);
            linea = leer.ReadLine();
        }
        leer.Close();
        SetUTCcorregido(); //Coregimos la hora para cambio de dia
```

```
            SetVelocidadDGPS();//Asignamos velocidades a los mensajes de
DGPS
            SetZonas();//Asignamos la zona en funcion de la posicion en el mapa
        }

        public void OrdenarPaquetesPorTiempo() //Te ordena de menor a mayor
por tiempo. A veces algun paquete tiene un tiempo anterior al anterior paquete
        {
            int i, j, k;
            DecodificadorMensaje temp;
            int indiceFinal = 0;
            int indiceInicial = 0;
            k = 1;
            bool encontrado = false;
            while ((k < myList.Count()) && (!encontrado))//Buscamos el cambio de
dia
            {
                if (myList[k - 1].GetUTC() > myList[k].GetUTC())
                {
                    //No podemos aplicar solo el criterio de mirar si el tiempo de un
paquete es menor al siguiente para encontrar el cambio de dia
                    //debido a que precisamente el problema es que algunos paquetes
no estan oredenados en tiempo.
                    encontrado = true;
                    int mMax = k - 1 + 100; //Comparamos el paquete con los
siguientes 100 paquetes. Asumimos que solo hay pequeños desordenes en la
lista, menores a 100 paquetes seguidos.
                    if (mMax >= myList.Count())
                    {
                        mMax = myList.Count() - 1;
                    }
                    for (int m = k + 1; m <= mMax; m++)
                    {
                        if (myList[k - 1].GetUTC() < myList[m].GetUTC())
                        {
                            encontrado = false;
                        }
                    }
                    if (encontrado)
                    {
                        indiceFinal = k - 1;
                    }
                }
                k++;
            }
            if ((!encontrado) || (indiceFinal == 0))
            {
                indiceFinal = myList.Count();
            }
```

```
        bool algoCambiado;//Si en algun momento para de cambiar, paramos el
bucle
        bool parar = false;//para el bucle
        if (indiceFinal != myList.Count())
        {
            i = indiceInicial + 1;
            while ((i <= indiceFinal) && (!parar))//Algoritmo que ordena de menor
a mayor (metodo burbuja, Bubble Sort)
            {
                algoCambiado = false;
                for (j = 0; j <= indiceFinal - 1; j++) //De menor a mayor
                {
                    if (myList[j].GetUTC() > myList[j + 1].GetUTC())//Ordenar de
menor a mayor teniendo en cuenta el cambio de dia
                    {
                        temp = myList[j];
                        myList[j] = myList[j + 1];
                        myList[j + 1] = temp;
                        algoCambiado = true;
                    }
                }
                if (!algoCambiado)
                {
                    parar = true;
                }
                i++;
            }
            indiceInicial = indiceFinal + 1;
            indiceFinal = myList.Count();
        }
        i = indiceInicial + 1; //Bubble empieza en 1. Ó indiceFinal +1 +1
        while ((i <= indiceFinal) && (!parar))//De menor a mayor
        {
            algoCambiado = false;
            for (j = 0; j <= indiceFinal - 1; j++) //De menor a mayor
            {
                if (myList[j].GetUTC() > myList[j + 1].GetUTC())
                {
                    temp = myList[j];
                    myList[j] = myList[j + 1];
                    myList[j + 1] = temp;
                    algoCambiado = true;
                }
            }
            if (!algoCambiado)
            {
                parar = true;
            }
            i++;
        }
```

```
        }

        public bool ComprobarDiferentesSIC() //Si devuelve true significa que hay
SIC=007(SMR) y =107(MLAT)
        {
            bool haySicDiferentes = false;
            int indice = 0;
            while ((myList[indice].GetSIC() == "No Data") && (indice <
myList.Count()))
            {
                indice++;
            }//Nos saltamos los NO Data
            if (indice < myList.Count())
            {
                int SIC = Convert.ToInt32(myList[indice].GetSIC());
                int i = indice + 1;
                while ((i < myList.Count()) && (!haySicDiferentes))
                {
                    if (SIC == 007)//Guardamos el primero y lo comparamos con el
resto
                    {
                        if (Convert.ToInt32(myList[i].GetSIC()) == 107)
                        {
                            haySicDiferentes = true;
                        }
                    }
                    if (SIC == 107)
                    {
                        if (Convert.ToInt32(myList[i].GetSIC()) == 007)
                        {
                            haySicDiferentes = true;
                        }
                    }
                    i++;
                }
            }
            return (haySicDiferentes);
        }

        public bool TodoSMR()//Comprueba si solo hay SMR en el fichero
        {
            bool todoSMR = true;
            int indice = 0;
            while ((myList[indice].GetSIC() == "No Data") && (indice <
myList.Count()))
            {
                indice++;
            }//Nos saltamos los NO Data
            if (indice < myList.Count())
            {
```

```
            while ((indice < myList.Count()) && (todoSMR))
            {
                if (Convert.ToInt32(myList[indice].GetSIC()) != 007)
                {
                    todoSMR = false;
                }
                indice++;
            }
        }
        return (todoSMR);
    }

    public LectorMensaje SepararSMRyADSB() //Si hay diferentes SIC solo se queda con uno de ellos (MLAT SIC 107)
    {
        LectorMensaje listaFinal = new LectorMensaje();
        for (int i = 0; i < myList.Count(); i++)
        {
            if (Convert.ToInt32(myList[i].GetSIC()) == 107)
            {
                listaFinal.AddPlanI(myList[i]);
            }
        }
        return (listaFinal);
    }

    public List<string> LeerVehiculosSquitter()//Lista con los ICAO Address de los vehiculos descartados
    {
        List<string> descartados = new List<string>();
        StreamReader leer = new StreamReader("VehiculosAEliminar.txt");
        string linea = leer.ReadLine();
        while (linea != null)
        {
            descartados.Add(linea);
            linea = leer.ReadLine();
        }
        leer.Close();
        return (descartados);
    }

    public LectorMensaje DescartarVehiculosSquitter() //Elimina los vehiculos que no envian señal de una manera normal
    {
        List<string> descartados = LeerVehiculosSquitter(); //Lista que contiene ICAOAdress de los vehiculos decartados
        LectorMensaje listaFinal = new LectorMensaje();
        bool descartar; //Devuelve true si el mensaje se debe descartar
        int i;
        for (int j = 0; j < myList.Count; j++)
```

```
        {
            descartar = false;
            i = 0;
            while ((i < descartados.Count()) && (!descartar))
            {
                if (myList[j].GetICAOAdress() != "No Data") //Evitamos que
compare con los No Data para reducir tiempo de procesado
                {
                    if (myList[j].GetICAOAdress() == descartados[i])
                    {
                        descartar = true;
                    }
                }
                i++;
            }
            if (!descartar)
            {
                listaFinal.AddPlanI(myList[j]);
            }
        }
        return (listaFinal);
    }

    public int SetIndicesDGPS(LectorMensaje lista)
    {
        //Al aplicar esta funcion a una lista de DGPS, la compara con la lista
introducida como parametro, y rellena el campo indiceDGPS de la lista de
DGPS
        //El campo indiceDGPS asigna la posicion en el vector myList en la que
se encuentra el punto con el que se va a comparar
        //Para escoger estos indices. Se busca en MyList los paquetes que
tienen el mismo ICAO Address que el D-GPS
        //Despues se mira que franja de tiempo que comparten y se fija un
tiempo inicial y final
        //Finalmente se escoge que puntos de la lista D-GPS estan mas cerca
(en tiempo) de la lista principal. En principio se tendra 1 dato/s de MLAT y
varios datos/s de D-GPS
        //Tambien retorna el numero de puntos emparejados

        for (int m = 0; m < myList.Count(); m++)
        {
            myList[m].SetIndiceDGPS(-1);//Puede que la lista se este recargando
(reasignando indices), por lo que reseteamos todo a -1
        }

        int match = 0;
        List<int> icaoList = ICAODetectedDGPS(lista);
        int indiceInicialMLAT = -1;
        int indiceFinalMLAT = -1;
        bool yaEsta = false;
```

```
            int i = 0;
            if (icaoList.Count() != 0)
            {
                if (myList[0].GetUTCcorregido() >=
lista.GetPlanI(icaoList[0]).GetUTCcorregido())
                {
                    if (myList[myList.Count() - 1].GetUTCcorregido() >=
lista.GetPlanI(icaoList[icaoList.Count() - 1]).GetUTCcorregido())
                    {
                        //Caso 1: D-GPS marca el inicio y MLAT el final (D-GPS
empieza despues de MLAT, D-GPS acaba despues de MLAT)
                        indiceFinalMLAT = icaoList.Count() - 1;
                        while ((i < icaoList.Count()) && (!yaEsta))
                        {
                            if (lista.GetPlanI(icaoList[i]).GetUTCcorregido() >=
myList[0].GetUTCcorregido())
                            {
                                indiceInicialMLAT = i;
                                yaEsta = true;
                            }
                            i++;
                        }
                    }
                    else
                    {
                        //Caso 2: D-GPS marca el inicio y el final
                        while ((i < icaoList.Count()) && (!yaEsta))
                        {
                            if (lista.GetPlanI(icaoList[i]).GetUTCcorregido() >=
myList[0].GetUTCcorregido())
                            {
                                indiceInicialMLAT = i;
                                yaEsta = true;
                            }
                            i++;
                        }
                        yaEsta = false;
                        i = 0;
                        while ((i < icaoList.Count()) && (!yaEsta))
                        {
                            if (lista.GetPlanI(icaoList[i]).GetUTCcorregido() >
myList[myList.Count() - 1].GetUTCcorregido())
                            {
                                indiceFinalMLAT = i - 1;//Nos pasamos con el >,
compensamos con -1
                                yaEsta = true;
                            }
                            i++;
                        }
```

```
            }
        }z
        else if (myList[0].GetUTCcorregido() <
lista.GetPlanI(icaoList[0]).GetUTCcorregido())
        {
            if (myList[myList.Count() - 1].GetUTCcorregido() <=
lista.GetPlanI(icaoList[icaoList.Count() - 1]).GetUTCcorregido())
            {
                //Caso 3: MLAT marca el inicio y D-GPS el final
                indiceInicialMLAT = 0;
                while ((i < icaoList.Count()) && (!yaEsta))
                {
                    if (lista.GetPlanI(icaoList[i]).GetUTCcorregido() >
myList[myList.Count() - 1].GetUTCcorregido())
                    {
                        indiceFinalMLAT = i - 1;//Nos pasamos con el >,
compensamos con -1
                        yaEsta = true;
                    }
                    i++;
                }
            }
            else
            {
                //Caso 4: MLAT marca el inicio y el final
                indiceInicialMLAT = 0;
                indiceFinalMLAT = icaoList.Count() - 1;
            }
        }
        else
        {
            //Caso 5: No se comparte franja de tiempo
            indiceInicialMLAT = -1;
            indiceFinalMLAT = -1;
        }

        int indiceI = Convert.ToInt32(myList.Count()/2); //En algunos ficheros
de D-GPS los primeros paquetes llegan muy separados en tiempo. Buscamos
la tasa de refresco en la mitad del fichero
        double tasaRefresco =
Math.Round((myList[indiceI+1].GetUTCcorregido() -
myList[indiceI].GetUTCcorregido()) * 60, 1);//solo se comparan puntos a +/- la
tasa de refresco
        if ((indiceInicialMLAT != -1) && (indiceFinalMLAT != -1))//Si hay franja
compartida
        {
            double diferenciaTiempoGanador;
            int indiceGanador;
            bool encontrado;
```

```
            int m = 1;//Esta fuera del for para que no se reinicie para cada
punto
            int guardarIndice;//Si un punto no se encuentra, el siguiente
empieza a analizar desde guardarIndice
            for (int n = indiceInicialMLAT; n <= indiceFinalMLAT; n++)
            {
                diferenciaTiempoGanador = Math.Pow(10, 8);
                indiceGanador = -1;
                encontrado = false;
                m = m - 1;
                guardarIndice = m;
                while ((m < myList.Count()) && (!encontrado))
                {
                    double diferenciaTemporalTiempo =
Math.Abs(myList[m].GetUTCcorregido() -
lista.GetPlanI(icaoList[n]).GetUTCcorregido()) * 60;
                    if (diferenciaTemporalTiempo <= tasaRefresco)
                    {
                        if (diferenciaTemporalTiempo < diferenciaTiempoGanador)
                        {
                            diferenciaTiempoGanador =
Math.Abs(myList[m].GetUTCcorregido() -
lista.GetPlanI(icaoList[n]).GetUTCcorregido()) * 60;
                            indiceGanador = m;
                        }
                    }
                    else
                    {
                        if (diferenciaTiempoGanador != Math.Pow(10, 8))
                        {
                            encontrado = true;//Si ya ha encontrado un punto
ganador, paramos de analizar. En el siguiente punto seguimos desde este
indice
                            //Si no se hace asi, el tiempo de procesado augmenta
considerablemente
                        }
                    }

                    m++;

                    if (m == myList.Count())
                    {
                        m = guardarIndice;
                        encontrado = true;//No se ha encontrado, pero asi se sale
del bucle
                    }
                }
                if (indiceGanador != -1)
                {
                    myList[indiceGanador].SetIndiceDGPS(icaoList[n]);
```

```
                    match++;
                }
            }
        }
    }
    return (match);
}

public List<int> ICAODetectedDGPS(LectorMensaje lista) //Retorna un vector que en cada posicion tiene los indices de myList que corresponden a un unico avion (compara ICAO Adress)
{
    //Funciona para listas de tipo D-GPS, necesita como parametro de entrada la lista general y usa Icao Adress del D-GPS

    int indice = 0;
    List<int> listaIcao = new List<int>();

    while ((lista.GetPlanI(indice).GetICAOAdress() == "No Data") && (indice < lista.GetNumList()))
    {
        indice++;
    }

    if (indice < lista.GetNumList())
    {
        for (int i = indice; i < lista.GetNumList(); i++)
        {
            if (lista.GetPlanI(i).GetICAOAdress() == myList[0].GetICAOAdress())
            {
                listaIcao.Add(i);
            }
        }
    }
    return (listaIcao);
}

public void SetZonas()
{
    //Determina a que zona pertenece cada punto recibido
    //Las zonas son una composicion de puntos que forman una zona cerrada.
    //El contorno de esta zona cerrada debe tener sentido antihorario.
    //Las zonas se comprueban en el codigo con el siguiente orden de prioridad: runway, stand, apron, taxi y airborne(el resto).
    //Esto permite, por ej., que apron tenga zonas superpuestas a stand, para reducir numero y complejidad de cada zona
    //Cada punto se somete por orden a cada zona a traves de un metodo de optimización lineal
```

```
        //Si un punto cumple en una zona, ya nos comprueba en la siguiente
        //Las zonas asignadas son las siguientes: -1/Sin zona (no hay datos de
posicion); 0/airborne; 01/type 4; 02/type 5
        // 11/pista 25L; 12/pista 02; 13/pista 25R;
        // 21/stand T1; 22/stand T2;  31/apron T1; 32/apron T2; 4/taxi

        bool[] puntoEncontrado = new bool[myList.Count()]; //false by default. Si
el punto se encuentra en una zona, ya no se compara en las siguientes zonas
        bool[] puntoNoCumple; //Si en una inecuacion se encuentra que no
cumple, ya no se compara con las siguiente inecuaciones de esa seccion
        double[,] posicion = new double[myList.Count(), 2];
        double[] velocidad = new double[myList.Count()]; //Velocidad absoluta
[m/s]

        for (int i = 0; i < myList.Count(); i++)
        {
            //Miramos si los datos de posicion del paquete vienen en WGS, del
SMR, o en coordenadas cartesianas del MLAT
            posicion[i, 0] = myList[i].GetPosicion()[0];
            posicion[i, 1] = myList[i].GetPosicion()[1];
            if ((posicion[i, 0] == Math.Pow(10, 8)) && (posicion[i, 1] ==
Math.Pow(10, 8)))
            {
                puntoEncontrado[i] = true; //Ya no entra en el codigo, zona se
queda en -1
            }

            if ((myList[i].GetCartTrackVelX() != Math.Pow(10, 8)) &&
(myList[i].GetCartTrackVelY() != Math.Pow(10, 8)))
            {
                velocidad[i] = Math.Sqrt(Math.Pow(myList[i].GetCartTrackVelX(), 2)
+ Math.Pow(myList[i].GetCartTrackVelY(), 2));
            }
            else if ((myList[i].GetPolarTrackVelV() != Math.Pow(10, 8)) &&
(myList[i].GetPolarTrackVelAngle() != Math.Pow(10, 8)))
            {
                velocidad[i] = myList[i].GetPolarTrackVelV();
            }
            else velocidad[i] = Math.Pow(10, 8);
        }

        StreamReader leer = new StreamReader("Secciones.txt");
        string linea1 = leer.ReadLine();
        string lineaInicial = linea1; //Guardamos el primer punto para poder
cerrar el controno con el ultimo punto
        string linea2 = leer.ReadLine();
        int numZona = 1;//Numero arbitrario asignado de manera ascendente a
cada zona de secciones.txt.
        bool finSeccion;//Controla si es seccion ha acabado
        bool ultimaLinea;//Se activa para comparar la ultima linea con la primera
```

```
        int velocidadLimitePista = 120; //[m/s]Las tres primeras zonas son las pistas. Las siguientes tendran velocidad limite de 35m/s
        int velocidadLimiteRodadura = 35; //Velocidad limite para filtrar aviones que sobrevuelan el aeropuerto.
        while (linea2 != null)
        {
            ultimaLinea = false;
            finSeccion = false;
            puntoNoCumple = new bool[myList.Count()];
            while (!finSeccion)
            {
                string[] punto = linea1.Split(' ');
                double[] coordenadaInicial = { Convert.ToDouble(punto[0]), Convert.ToDouble(punto[1]) };
                punto = linea2.Split(' ');
                double[] coordenadaFinal = { Convert.ToDouble(punto[0]), Convert.ToDouble(punto[1]) };
                double Ax = coordenadaFinal[0] - coordenadaInicial[0];
                double Ay = coordenadaFinal[1] - coordenadaInicial[1];
                string operador = "<"; //Determina el sentido de la inecuacion. Premisa: contorno de la seccion antihorario
                if (((Ax > 0) && (Ay >= 0)) || ((Ax >= 0) && (Ay < 0)))
                    operador = ">";
                //Sino (((Ax <= 0) && (Ay > 0)) || ((Ax < 0) && (Ay <= 0))) por lo que operador permanece = "<";

                if (operador == ">") //Si el punto no cumple alguna de las inecuaciones de la seccion, no cumple
                {
                    for (int i = 0; i < myList.Count(); i++)
                    {
                        if ((!puntoEncontrado[i]) && (!puntoNoCumple[i]))
                        {
                            if (posicion[i, 1] - (Ay / Ax) * posicion[i, 0] < coordenadaInicial[1] - (Ay / Ax) * coordenadaInicial[0])
                                puntoNoCumple[i] = true;
                        }
                    }
                }
                else
                {
                    for (int i = 0; i < myList.Count(); i++)
                    {
                        if ((!puntoEncontrado[i]) && (!puntoNoCumple[i]))
                        {
                            if (posicion[i, 1] - (Ay / Ax) * posicion[i, 0] > coordenadaInicial[1] - (Ay / Ax) * coordenadaInicial[0])
                                puntoNoCumple[i] = true;
                        }
                    }
```

```
                }

                linea1 = linea2;
                linea2 = leer.ReadLine();
                if (ultimaLinea)
                    finSeccion = true;
                if (linea2 == "/")
                {
                    ultimaLinea = true;
                    linea2 = lineaInicial;
                }
            }
            for (int i = 0; i < myList.Count(); i++)
            {
                if ((!puntoNoCumple[i]) && (!puntoEncontrado[i]))
                {
                    puntoEncontrado[i] = true;
                    if (numZona <= 5)//Las 5 primeras zonas son las 3 pistas y los
dos cruces
                    {
                        if (velocidad[i] <= velocidadLimitePista)
                        {
                            if ((numZona == 1) || (numZona == 3))// cruce de pista 25L
con aproximacion a 02 o 25R con 02
                            {
                                double anguloVelocidad; //Angulo de movimiento [º]
                                if ((myList[i].GetCartTrackVelX() != Math.Pow(10, 8)) &&
(myList[i].GetCartTrackVelY() != Math.Pow(10, 8)))
                                {
                                    if ((myList[i].GetCartTrackVelX() == 0) &&
(myList[i].GetCartTrackVelY() > 0))
                                        anguloVelocidad = 90;
                                    else if ((myList[i].GetCartTrackVelX() == 0) &&
(myList[i].GetCartTrackVelY() < 0))
                                        anguloVelocidad = 270;
                                    else if ((myList[i].GetCartTrackVelX() > 0) &&
(myList[i].GetCartTrackVelY() == 0))
                                        anguloVelocidad = 0;
                                    else if ((myList[i].GetCartTrackVelX() < 0) &&
(myList[i].GetCartTrackVelY() == 0))
                                        anguloVelocidad = 180;
                                    else if (myList[i].GetCartTrackVelX() < 0)
                                        anguloVelocidad =
Math.Atan(myList[i].GetCartTrackVelY() / myList[i].GetCartTrackVelX()) * (180 /
Math.PI) + 180;
                                    else if ((myList[i].GetCartTrackVelX() > 0) &&
(myList[i].GetCartTrackVelY() < 0))
                                        anguloVelocidad =
Math.Atan(myList[i].GetCartTrackVelY() / myList[i].GetCartTrackVelX()) * (180 /
Math.PI) + 360;
```

```
                        else anguloVelocidad =
Math.Atan(myList[i].GetCartTrackVelY() / myList[i].GetCartTrackVelX()) * (180 /
Math.PI);

                        if (((anguloVelocidad >= 47.75) && (anguloVelocidad
<= 94.25)) || ((anguloVelocidad >= 227.25) && (anguloVelocidad <= 274.25)))
                        {
                            if (numZona == 1)
                                myList[i].SetZona(01);//Sigue en airborne,
haciendo SID/STAR, Area tipo 4
                        }
                        else
                        {
                            if (numZona == 3)
                                myList[i].SetZona(13);
                        }
                    }
                    else if ((myList[i].GetPolarTrackVelV() != Math.Pow(10,
8)) && (myList[i].GetPolarTrackVelAngle() != Math.Pow(10, 8)))
                    {
                        anguloVelocidad = myList[i].GetPolarTrackVelAngle();
                        if ((anguloVelocidad <= 42.25) || (anguloVelocidad >=
355.75) || ((anguloVelocidad >= 222.25) && (anguloVelocidad <= 175.75)))
                        {
                            if (numZona == 1)
                                myList[i].SetZona(01);
                        }
                        else
                        {
                            if (numZona == 3)
                                myList[i].SetZona(13);
                        }
                    }
                    if (numZona == 1)
                    {
                        if (myList[i].GetZona() != 01)
                            puntoEncontrado[i] = false;
                    }
                    if (numZona == 3)
                    {
                        if (myList[i].GetZona() != 13)
                            puntoEncontrado[i] = false;
                    }
                }
                else if (numZona == 2)//11 pista 25L
                    myList[i].SetZona(11);
                else if (numZona == 4)//12 pista 02
                    myList[i].SetZona(12);
                else if (numZona == 5)//13 pista 25R
                    myList[i].SetZona(13);
```

```
                    }
                    else myList[i].SetZona(0); //Zona Airborne
                }
                else
                {
                    if (velocidad[i] <= velocidadLimiteRodadura)
                    {
                        if ((numZona >= 6) && (numZona <= 9))//21 stand T1
                            myList[i].SetZona(21);
                        else if ((numZona >= 10) && (numZona <= 16))//22 stand
T2
                            myList[i].SetZona(22);
                        else if ((numZona >= 17) && (numZona <= 22))//31 apron
T1
                            myList[i].SetZona(31);
                        else if ((numZona >= 23) && (numZona <= 25))//32 apron
T2
                            myList[i].SetZona(32);
                        else if ((numZona >= 26) && (numZona <= 36))//4 taxi
                            myList[i].SetZona(4);
                        else
                            myList[i].SetZona(0);
                    }
                    else //Si la vel no es superior a la limite de rodadura no entra
aqui, descartando posibles blancos fijos dentro de las areas 4,5
                    {
                        if ((numZona >= 37) && (numZona <= 42))//01 Area tipo 4
                            myList[i].SetZona(01);
                        else if ((numZona >= 43) && (numZona <= 48))//02 Area
tipo 5
                            myList[i].SetZona(02);
                        else
                            puntoEncontrado[i] = false;
                    }
                }
            }
        }
        if (linea2 != null)
        {
            linea1 = linea2;
            lineaInicial = linea1;
            linea2 = leer.ReadLine();
        }
        numZona++;
    }
    for (int i = 0; i < myList.Count(); i++)
    {
        if (!puntoEncontrado[i])
        {
```

```
                myList[i].SetZona(0);//Su posicion es conocida, pero no esta en
ninguna area definida. Tampoco sobrevuela LEBL
            }
        }
        leer.Close();
    }

    public void SetUTCcorregido()//A las horas de >24 les suma 24
    {
        if (myList.Count() != 0)
        {
            bool cambioDia = false;
            myList[0].SetUTCcorregido(myList[0].GetUTC());
            for (int i = 1; i < myList.Count(); i++)
            {
                if (!cambioDia)
                {
                    if (myList[i].GetUTC() < myList[i - 1].GetUTC())
                    {
                        myList[i].SetUTCcorregido(myList[i].GetUTC() + 24 * 60);
                        cambioDia = true;
                    }
                    else
                    {
                        myList[i].SetUTCcorregido(myList[i].GetUTC());
                    }
                }
                else myList[i].SetUTCcorregido(myList[i].GetUTC() + 24 * 60);
            }
        }
    }

    public void SetVelocidadDGPS() //Asignamos velocidad para poder
despues asignar una zona
    {
        for (int i = 0; i < myList.Count() - 1; i++)
        {
            double[] posI = myList[i].GetPosicion();
            double tiempoI = myList[i].GetUTCcorregido() * 60;
            double[] posF = myList[i + 1].GetPosicion();
            double tiempoF = myList[i + 1].GetUTCcorregido() * 60;
            if (tiempoF - tiempoI != 0)
            {
                myList[i].SetVelocidad((posF[0] - posI[0]) / (tiempoF - tiempoI),
(posF[1] - posI[1]) / (tiempoF - tiempoI));
            }
        }
        //La ultima velocidad es igual a la penultima (aproximación)
        myList[myList.Count() - 1].SetVelocidad(myList[myList.Count() -
2].GetCartTrackVelX(), myList[myList.Count() - 2].GetCartTrackVelY());
```

```
        }

        public List<List<int>> AircraftICAODetected() //Retorna una matriz que en
cada fila tiene los indices de myList que corresponden a un unico avion
(compara ICAO Adress)
        {
            int indice = 0;
            List<string> listaAviones = new List<string>();

            while ((myList[indice].GetICAOAdress() == "No Data") && (indice <
myList.Count()))
            {
                indice++;
            }

            if (indice < myList.Count())
            {
                listaAviones.Add(myList[indice].GetICAOAdress());
                for (int i = indice + 1; i < myList.Count(); i++)
                {
                    if (myList[i].GetICAOAdress() != "No Data")
                    {
                        bool yaExiste = false;
                        int n = 0;
                        while ((n < listaAviones.Count()) && (!yaExiste))
                        {
                            if (myList[i].GetICAOAdress() == listaAviones[n])
                            {
                                yaExiste = true;
                            }
                            else n++;
                        }
                        if (!yaExiste)
                            listaAviones.Add(myList[i].GetICAOAdress());
                    }
                }
            }

            List<List<int>> matrizAviones = new List<List<int>>();
            for (int i = 0; i < listaAviones.Count(); i++)
            {
                List<int> filaAviones = new List<int>();
                for (int n = 0; n < myList.Count(); n++)
                {
                    if (listaAviones[i] == myList[n].GetICAOAdress())
                    {
                        filaAviones.Add(n);
                    }
                }
                matrizAviones.Add(filaAviones);
```

```
        }
        return (matrizAviones);
    }

    public List<List<int>> AircraftTrackNumberDetected() //Retorna una matriz que en cada fila tiene los indices de myList que corresponden a un unico avion (compara numero de pista)
    {
        int indice = 0;
        List<string> listaAviones = new List<string>();

        while ((myList[indice].GetTrackNumber() == "No Data") && (indice < myList.Count()))
        {
            indice++;
        }

        if (indice < myList.Count())
        {
            listaAviones.Add(myList[indice].GetTrackNumber());
            for (int i = indice + 1; i < myList.Count(); i++)
            {
                if (myList[i].GetTrackNumber() != "No Data")
                {
                    bool yaExiste = false;
                    int n = 0;
                    while ((n < listaAviones.Count()) && (!yaExiste))
                    {
                        if (myList[i].GetTrackNumber() == listaAviones[n])
                        {
                            yaExiste = true;
                        }
                        else n++;
                    }
                    if (!yaExiste)
                        listaAviones.Add(myList[i].GetTrackNumber());
                }
            }
        }

        List<List<int>> matrizAviones = new List<List<int>>();
        for (int i = 0; i < listaAviones.Count(); i++)
        {
            List<int> filaAviones = new List<int>();
            for (int n = 0; n < myList.Count(); n++)
            {
                if (listaAviones[i] == myList[n].GetTrackNumber())
                {
                    filaAviones.Add(n);
                }
```

```
            }
            matrizAviones.Add(filaAviones);
        }
        return (matrizAviones);
    }

    public double[,] GetDimensiones(string icao)//Compara las dimensiones de los diferente puntos de un vehiculo
    {
        double[,] dimensiones = new double[2, 2];
        double[,] dimensionesi = new double[2, 2];
        if (icao != "")
        {
            int i = 0;
            bool encontrado = false;
            while ((i < myList.Count())&&(!encontrado))
            {
                if(myList[i].GetICAOAdress()==icao)
                {
                    encontrado = true;
                    dimensiones[0, 0] = Convert.ToInt32(myList[i].GetPosicion()[0]);
                    dimensiones[0, 1] = Convert.ToInt32(myList[i].GetPosicion()[0]);
                    dimensiones[1, 0] = Convert.ToInt32(myList[i].GetPosicion()[1]);
                    dimensiones[1, 1] = Convert.ToInt32(myList[i].GetPosicion()[1]);
                }
                i++;
            }

            while (i < myList.Count())
            {
                if (myList[i].GetICAOAdress() == icao)
                {
                    dimensionesi[0, 0] = Convert.ToInt32(myList[i].GetPosicion()[0]);
                    dimensionesi[0, 1] = Convert.ToInt32(myList[i].GetPosicion()[0]);
                    dimensionesi[1, 0] = Convert.ToInt32(myList[i].GetPosicion()[1]);
                    dimensionesi[1, 1] = Convert.ToInt32(myList[i].GetPosicion()[1]);
                    if (dimensiones[0, 0] > dimensionesi[0, 0])
                    {
                        dimensiones[0, 0] = dimensionesi[0, 0];
                    }
                    if (dimensiones[0, 1] < dimensionesi[0, 1])
                    {
                        dimensiones[0, 1] = dimensionesi[0, 1];
                    }
                    if (dimensiones[1, 0] > dimensionesi[1, 0])
                    {
                        dimensiones[1, 0] = dimensionesi[1, 0];
                    }
                    if (dimensiones[1, 1] < dimensionesi[1, 1])
                    {
```

```
                    dimensiones[1, 1] = dimensionesi[1, 1];
                }
            }
            i++;
        }
    }
    return (dimensiones);
}

public double GetProporcion(double[,] dimensiones, double
panelDimensionesHeight, double panelDimensionesWidth)//Valor con el que
proporcionar el mapa
{
    double ProporcionY = Math.Abs((dimensiones[1, 1] - dimensiones[1, 0]))
/ panelDimensionesHeight;
    double ProporcionX = Math.Abs(dimensiones[0, 1] - dimensiones[0, 0]) /
panelDimensionesWidth;
    double Proporcion;
    if (ProporcionX > ProporcionY)
    {
        Proporcion = ProporcionX;
    }
    else Proporcion = ProporcionY;

    return (Proporcion);
}
}
}
```

CMap.cs

```
using System;
using System.IO;

namespace Codigo
{
   public class CMap
   {
      double[,] cartLine;
      double[,] cartPoli;
      string[] Mlat = new string[] { "411749426N", "0020442410E"
};//Coordenadas del mlat y SMR
      string[] SMR = new string[] { "411744226N", "0020542411E" };
      double[] cartMlat = new double[2];
      double[] cartSMR = new double[2];

      string nombreFichero;

      public CMap(string NombreFichero)
      {
         this.nombreFichero = NombreFichero;
      }

      public int[] EncontrarLongitud()//Encuentra la longitud de lineas y polilineas
para la función de leer()
      {
         StreamReader leer = new StreamReader(nombreFichero + ".txt");
         string linea = leer.ReadLine();
         int contadorLinea = 0;
         int contadorPoli = 0;
         bool llegarPoli = false;
         while (linea != "")
         {
            if (llegarPoli == false)
            {
               if (linea.Contains("Polilinea") == false)
               {
                  contadorLinea = contadorLinea + 1;
               }
               else
               {
                  llegarPoli = true;
               }
            }
            else
            {
               contadorPoli = contadorPoli + 1;
            }
            linea = leer.ReadLine();
```

```
        }

        leer.Close();
        int[] result = new int[2];
        result[0] = contadorLinea;
        result[1] = contadorPoli;
        return (result);
    }

    public void Leer()
    {
        StreamReader leer = new StreamReader(nombreFichero + ".txt");
        string linea = leer.ReadLine();
        int[] longitud = EncontrarLongitud();
        cartLine = new double[longitud[0], 4];
        cartPoli = new double[longitud[1], 2];

        bool llegarPoli = false;
        cartMlat[0] = ConvertWGStoRad(Mlat[0], 0);//Mlat en radianes
        cartMlat[1] = ConvertWGStoRad(Mlat[1], 1);
        int i = 0; //fila de la matriz
        while (linea != "")
        {
            if (llegarPoli == false)
            {
                if (linea.Contains("Polilinea") == false)
                {
                    string[] entradas = linea.Split(' ');
                    //entradas contiene ej:line 454545555N(lat,y)
0454545555E(long,x) 454545555N 0454545555E
                    cartLine[i, 0] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(entradas[1], 0), ConvertWGStoRad(entradas[2], 1))[0]);
                    cartLine[i, 1] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(entradas[1], 0), ConvertWGStoRad(entradas[2], 1))[1]);
                    cartLine[i, 2] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(entradas[3], 0), ConvertWGStoRad(entradas[4], 1))[0]);
                    cartLine[i, 3] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(entradas[3], 0), ConvertWGStoRad(entradas[4], 1))[1]);
                    i = i + 1;
                }
                else
                {
                    llegarPoli = true;
                    i = 0;
                }
            }
            else
            {
                if (linea.Contains("Polilinea") == false)
                {
```

```
                    string[] poliEntradas = linea.Split(' ');
                    cartPoli[i, 0] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(poliEntradas[0], 0), ConvertWGStoRad(poliEntradas[1],
1))[0]);
                    cartPoli[i, 1] = Convert.ToInt32(ConvertRadtoXY(cartMlat,
ConvertWGStoRad(poliEntradas[0], 0), ConvertWGStoRad(poliEntradas[1],
1))[1]);
                    i = i + 1;
                }
                else
                {
                    cartPoli[i, 0] = Math.Pow(10, 8); //ponemos una linea con
numero imposible para ver el salto entre polilineas, excepto la primera polilinea
                    cartPoli[i, 1] = 0; //con que la primera componente sea grande ya
lo detectamos, asi ocupa menos. CREO
                    i = i + 1;
                }
            }
            linea = leer.ReadLine();
        }
    }

    public double ConvertWGStoRad(string coordenada, int orientacion)//Si es
lat, orientacion=0; Si es long, orientacion=1. El nombre expresa mal su funcion.
Pasa de lat long a radianes
    {
        //Retorna una coordenada en radianes
        double grados = 0;
        int latSigno = 1; //por defecto es positivo para Norte
        int longSigno = 1; //por defecto es positivo para Este
        if (orientacion == 0)
        {
            if (Convert.ToString(coordenada[9]) == "S")
            {
                latSigno = -1;
            }
            grados = Convert.ToDouble(Convert.ToString(coordenada[0]) +
Convert.ToString(coordenada[1]));
            double minutos =
Convert.ToDouble(Convert.ToString(coordenada[2]) +
Convert.ToString(coordenada[3]));
            double segundos =
Convert.ToDouble(Convert.ToString(coordenada[4]) +
Convert.ToString(coordenada[5]));
            double decimas =
Convert.ToDouble(Convert.ToString(coordenada[6]) +
Convert.ToString(coordenada[7]) + Convert.ToString(coordenada[8])) / 1000;
            segundos = segundos + decimas;
            grados = grados + (minutos / 60) + (segundos / 3600);
            grados = latSigno * grados * Math.PI / 180;
```

```
            }
            if (orientacion == 1)
            {
                if ((Convert.ToString(coordenada[10]) == "W") ||
(Convert.ToString(coordenada[10]) == "O"))
                {
                    longSigno = -1;
                }
                grados = Convert.ToDouble(Convert.ToString(coordenada[0]) +
Convert.ToString(coordenada[1]) + Convert.ToString(coordenada[2]));
                double minutos =
Convert.ToDouble(Convert.ToString(coordenada[3]) +
Convert.ToString(coordenada[4]));
                double segundos =
Convert.ToDouble(Convert.ToString(coordenada[5]) +
Convert.ToString(coordenada[6]));
                double decimas =
Convert.ToDouble(Convert.ToString(coordenada[7]) +
Convert.ToString(coordenada[8]) + Convert.ToString(coordenada[9])) / 1000;
                segundos = segundos + decimas;
                grados = grados + (minutos / 60) + (segundos / 3600);
                grados = longSigno * grados * Math.PI / 180;

            }
            return (grados);
        }

        public double[] ConvertRadtoXY(double[] llh0, double dphi, double
dlam)//proyecta WGS84 sobre el plano. Lat long en radianes
        {
            double a = 6378137;
            double b = 6356752.3142;
            double e2;
            e2 = 1 - Math.Pow((b / a), 2);
            //Location of reference point in radians
            double phi = llh0[0];//lat
            double lam = llh0[1];//long
            //Some useful definitions
            double tmp1 = Math.Sqrt(1 - e2 * Math.Pow(Math.Sin(phi), 2));
            double cl = Math.Cos(lam);
            double sl = Math.Sin(lam);
            double cp = Math.Cos(phi);
            double sp = Math.Sin(phi);
            //Location of data points in radians
            double[] denu = new double[2];
            dphi = dphi - phi;
            dlam = dlam - lam;
            //Transformations
            denu[0] = (a / tmp1) * cp * dlam - (a * (1 - e2) / (Math.Pow(tmp1, 3))) * sp
* dphi * dlam; //X
```

```
            denu[1] = (a * (1 - e2) / Math.Pow(tmp1, 3)) * dphi + 1.5 * cp * sp * a * e2
* Math.Pow(dphi, 2) + 0.5 * sp * cp * (a / tmp1) * Math.Pow(dlam, 2); //Y

            return (denu);
        }

        public double[,] GetDimensiones(double[,] Lvector, double[,]
Pvector)//Obtiene los limites laterales del vector para plotearlo
        {
            //Compara el vector de polilineas con el vector de lineas
            double[,] dimensiones = new double[2, 2];

            int Prows = Pvector.GetLength(0);
            int Pcolumns = Pvector.GetLength(1);

            double Pxm = Pvector[0, 0];
            double PxM = Pvector[0, 0];
            double Pym = Pvector[0, 1];
            double PyM = Pvector[0, 1];

            double xm;
            double xM;
            double ym;
            double yM;

            int Lrows = Lvector.GetLength(0);
            int Lcolumns = Lvector.GetLength(1);
            double Lxm = Lvector[0, 0];
            double LxM = Lvector[0, 0];
            double Lym = Lvector[0, 1];
            double LyM = Lvector[0, 1];
            if (Lcolumns == 4)
            {
                int cont = 0;
                while (cont < Lrows)
                {
                    if (Lvector[cont, 1] < Lym)
                    {
                        Lym = Lvector[cont, 1];
                    }
                    if (Lvector[cont, 0] < Lxm)
                    {
                        Lxm = Lvector[cont, 0];
                    }
                    if (Lvector[cont, 3] < Lym)
                    {
                        Lym = Lvector[cont, 3];
                    }
                    if (Lvector[cont, 2] < Lxm)
                    {
```

```
                Lxm = Lvector[cont, 2];
            }
            if (Lvector[cont, 1] > LyM)
            {
                LyM = Lvector[cont, 1];
            }
            if (Lvector[cont, 0] > LxM)
            {
                LxM = Lvector[cont, 0];
            }
            if (Lvector[cont, 3] > LyM)
            {
                LyM = Lvector[cont, 3];
            }
            if (Lvector[cont, 2] > LxM)
            {
                LxM = Lvector[cont, 2];
            }
            cont = cont + 1;
        }
    }
    if (Pcolumns == 2)
    {
        int cont = 0;
        while (cont < Prows)
        {
            if (Pvector[cont, 0] != Math.Pow(10, 8))
            {
                if (Pvector[cont, 1] < Pym)
                {
                    Pym = Pvector[cont, 1];
                }
                if (Pvector[cont, 0] < Pxm)
                {
                    Pxm = Pvector[cont, 0];
                }
                if (Pvector[cont, 1] > PyM)
                {
                    PyM = Pvector[cont, 1];
                }
                if (Pvector[cont, 0] > PxM)
                {
                    PxM = Pvector[cont, 0];
                }
            }
            cont = cont + 1;
        }
    }

    if (Lxm > Pxm)
```

```
            {
                xm = Pxm;
            }
            else xm = Lxm;
            if (Lym > Pym)
            {
                ym = Pym;
            }
            else ym = Lym;
            if (LxM < PxM)
            {
                xM = PxM;
            }
            else xM = LxM;
            if (LyM < PyM)
            {
                yM = PyM;
            }
            else yM = LyM;

            dimensiones[0, 0] = Convert.ToInt32(xm);
            dimensiones[0, 1] = Convert.ToInt32(xM);
            dimensiones[1, 0] = Convert.ToInt32(ym);
            dimensiones[1, 1] = Convert.ToInt32(yM);
            return (dimensiones);
        }

        public double GetProporcion(double[,] dimensiones, double
panelDimensionesHeight, double panelDimensionesWidth)//Valor con el que
proporcionalizar el mapa
        {
            double ProporcionY = Math.Abs((dimensiones[1, 1] - dimensiones[1, 0]))
/ panelDimensionesHeight;
            double ProporcionX = Math.Abs(dimensiones[0, 1] - dimensiones[0, 0]) /
panelDimensionesWidth;
            double Proporcion;
            if (ProporcionX > ProporcionY)
            {
                Proporcion = ProporcionX;
            }
            else Proporcion = ProporcionY;

            return (Proporcion);
        }

        public double[,] GetLinea()
        {
            return cartLine;
        }
```

```
        public double[,] GetPoli()
        {
            return cartPoli;
        }
    }
}
```

CListaMap.cs

```
using System.Collections.Generic;
using System.Linq;

namespace Codigo
{
    public class CListaMap
    {
        List<CMap> myList = new List<CMap>();

        public CListaMap()
        {
            myList.Add(new CMap("aeropuerto"));
            myList.Add(new CMap("Aeropuerto_Barcelona"));
            myList.Add(new CMap("Aeropuerto_Barcelonanue"));
            myList.Add(new CMap("BCN_Aparcamientos"));
            myList.Add(new CMap("BCN_CarreterasServicio"));
            myList.Add(new CMap("BCN_Edificios"));
            myList.Add(new CMap("BCN_Parterres"));
            myList.Add(new CMap("BCN_Pistas"));
            myList.Add(new CMap("BCN_ZonasMovimiento"));

            CargarMapas();
        }

        public CMap GetPlanI(int i)
        {
            return myList[i];
        }

        public void CargarMapas()
        {
            int i = 0;
            while (i < myList.Count)
            {
                myList[i].Leer();
                i++;
            }
        }
```

```
        public int GetNumList()
        {
            return myList.Count();
        }

        public double[,] GetDIMENSIONES()//Compara las dimensiones de todos
los mapas
        {
            int i = 1;
            double[,] dimensiones = new double[2, 2];
            double[,] dimensionesi = new double[2, 2];
            dimensiones = myList[0].GetDimensiones(myList[0].GetLinea(),
myList[0].GetPoli());
            while (i < myList.Count)
            {
                dimensionesi = myList[i].GetDimensiones(myList[i].GetLinea(),
myList[i].GetPoli());
                if (dimensiones[0, 0] > dimensionesi[0, 0])
                {
                    dimensiones[0, 0] = dimensionesi[0, 0];
                }
                if (dimensiones[0, 1] < dimensionesi[0, 1])
                {
                    dimensiones[0, 1] = dimensionesi[0, 1];
                }
                if (dimensiones[1, 0] > dimensionesi[1, 0])
                {
                    dimensiones[1, 0] = dimensionesi[1, 0];
                }
                if (dimensiones[1, 1] < dimensionesi[1, 1])
                {
                    dimensiones[1, 1] = dimensionesi[1, 1];
                }
                i++;
            }

            return (dimensiones);
        }

        public double GetPROPORCION(double PH, double PW)//Valor mas
grande de todos los mapas
        {
            int i = 1;
            double Proporcion;

            Proporcion =
myList[0].GetProporcion(myList[0].GetDimensiones(myList[0].GetLinea(),
myList[0].GetPoli()), PH, PW);
            while (i < myList.Count)
            {
```

```
            if (Proporcion <
myList[i].GetProporcion(myList[i].GetDimensiones(myList[i].GetLinea(),
myList[i].GetPoli()), PH, PW))
            {
                Proporcion =
myList[i].GetProporcion(myList[i].GetDimensiones(myList[i].GetLinea(),
myList[i].GetPoli()), PH, PW);
            }
            i = i + 1;
        }
        return (Proporcion);
    }
  }
}
```

Avaluador.cs

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace Codigo
{
   public class Avaluador
   {
      LectorMensaje myList;
      LectorMensaje DGPSList;
      LectorMensaje listaPA;
      double[,] updateRate; //Cada fila de la matriz corresponde a una zona (siguiente orden):
                            //maneouvering area; rwy 25L; rwy 02; rwy 25R; taxi; apron; apron T1; apron T2; stand; stand T1; stand T2; airborne; airborne STAR/SID; airborne Cruise
                            //maneouvering area, apron, stand y airborne son la media de las zonas que la componen.
                            //Cada columna de la matriz corresponde a: Updates, Expected update; update rate; minimum UR
      double[,] updatesUnitarios; //Lo calculamos para poder decidir si descartar o no un vehiculo
      double[,] PosAcc; //Filas ideanticas a updateRate sin desglosar Airborne (no data)
                        //Cada columna de la matriz corresponde a: P95, P99 ; Max Value detected; max P95; max P99; Max value allowed; Mean; Std Dev
      List<double[]> diferencias; //3 columnas: [0]= error en x, [1]=error en y; [2]=zona.
      double[,] MLATDet; //Filas ideanticas a updateRate sin desglosar Apron
                         //Cada columna de la matriz corresponde a: Detected, Expected ; PD; minimum PD
      double[,] Identification; //Filas ideanticas a updateRate + Total
                                //Cada columna de la matriz corresponde a: Correct; Incorrect; PID; minimum PID
      double[,] FalseDetection; //Filas ideanticas a updateRate + Total, Airborne solo tipo 4 y 5
                                //Cada columna de la matriz corresponde a: Reports, False ; PFD; minimum PFD
      double[,] FalseIdentification;//Filas ideanticas a updateRate + Total
                                    //Cada columna de la matriz corresponde a: Total; False; PFI; minimum PFI

      List<List<int>> aircraftDetected;//Matriz con los aviones estructurados segun su ICAO Adress
      List<List<int>> aircraftTrackDetected;//Matriz con los aviones estructurados segun su Track Number
```

```
        public Avaluador(LectorMensaje listaActual)//Contructor en caso de
analizar MLAT
        {
            this.myList = listaActual;
            aircraftDetected = myList.AircraftICAODetected();
            aircraftTrackDetected = myList.AircraftTrackNumberDetected();
            SetParameters("MLAT");
        }

        public Avaluador(LectorMensaje listaActual, LectorMensaje
listaDGPS)//Constructor en caso de analizar MLAT + D-GPS
        {
            this.DGPSList = listaDGPS;//se usa en AdaptarListaActual
            this.listaPA = listaActual;
            this.myList = AdaptarListaActual(listaActual);//Hace remove de todos los
paquetes que no son del vehiculo.
            aircraftDetected = myList.AircraftICAODetected();
            aircraftTrackDetected = myList.AircraftTrackNumberDetected();
            diferencias = SetDiferencias();
            SetParameters("DGPS");
        }

        public void SetParameters(string mode)
        {
            if (mode=="MLAT")
            {
                SetUpdateRate();
                SetProbOfMLATDetection();
                SetProbOfIdentification();
                SetProbOfFalseDetection();
                SetProbOfFalseIdentification();
            }
            else if (mode=="DGPS")
            {
                //Contiene PA, y ProbOfFalseDetDGPS
                SetUpdateRate();
                SetPositionAccuracy();
                SetProbOfMLATDetection();
                SetProbOfIdentification();
                SetProbOfFalseDetectionDGPS();
                SetProbOfFalseIdentification();
            }
        }

        public void SetUpdateRate()
        {
            updateRate = new double[15, 4];
            updatesUnitarios = new double[aircraftDetected.Count(), 3]; //0=real,
1=expected
```

```
        //Updates Reales
        bool tiempoSinSeñalSuperado = false;//Si tiempo actual es más grande
que el anterior +1min
        for (int i = 0; i < aircraftDetected.Count(); i++)
        {
            int zonaActual = myList.GetPlanI(aircraftDetected[i][0]).GetZona();
            int indiceInicial = 0;
            updatesUnitarios[i, 2] = aircraftDetected[i][0];//Guardamos el indice
para obtener icao Address
            for (int n = 1; n < aircraftDetected[i].Count(); n++)
            {
                if (myList.GetPlanI(aircraftDetected[i][n]).GetUTC() <
myList.GetPlanI(aircraftDetected[i][n - 1]).GetUTC())//Filtramos el caso de paso
de 24h a 0h
                {
                    if (myList.GetPlanI(aircraftDetected[i][n]).GetUTC() + 24 * 60 >
myList.GetPlanI(aircraftDetected[i][n - 1]).GetUTC() + 1)//Si tiempo actual es
más grande que el anterior +1min, dividimos en 2 el vuelo
                    {
                        tiempoSinSeñalSuperado = true;
                    }
                }
                else if (myList.GetPlanI(aircraftDetected[i][n]).GetUTC() >
myList.GetPlanI(aircraftDetected[i][n - 1]).GetUTC() + 1)//Si tiempo actual es
más grande que el anterior +1min, dividimos en 2 el vuelo
                {
                    tiempoSinSeñalSuperado = true;
                }

                if ((myList.GetPlanI(aircraftDetected[i][n]).GetZona() != zonaActual)
|| (n == aircraftDetected[i].Count() - 1) || (tiempoSinSeñalSuperado))
                {
                    tiempoSinSeñalSuperado = false;
                    int indiceFinal;
                    if (n != aircraftDetected[i].Count() - 1)
                        indiceFinal = n - 1;
                    else indiceFinal = n;

                    int updates = 1;
                    for (int m = indiceInicial + 1; m <= indiceFinal; m++)
                    {
                        double tiempoInicial = myList.GetPlanI(aircraftDetected[i][m -
1]).GetUTC() * 60;
                        double tiempoFinal =
myList.GetPlanI(aircraftDetected[i][m]).GetUTC() * 60;
                        if (tiempoFinal < tiempoInicial)
                        {
                            tiempoFinal = tiempoFinal + 24 * 3600;
                        }
                        if (tiempoFinal <= tiempoInicial+1)
```

```
        {
            updates++;
        }
    }

    if (zonaActual == 11)
    {
        updateRate[1, 0] = updateRate[1, 0] + updates;
    }
    else if (zonaActual == 12)
    {
        updateRate[2, 0] = updateRate[2, 0] + updates;
    }
    else if (zonaActual == 13)
    {
        updateRate[3, 0] = updateRate[3, 0] + updates;
    }
    else if (zonaActual == 4)
    {
        updateRate[4, 0] = updateRate[4, 0] + updates;
    }
    else if (zonaActual == 31)
    {
        updateRate[6, 0] = updateRate[6, 0] + updates;
    }
    else if (zonaActual == 32)
    {
        updateRate[7, 0] = updateRate[7, 0] + updates;
    }
    else if (zonaActual == 21)
    {
        updateRate[9, 0] = updateRate[9, 0] + updates;
    }
    else if (zonaActual == 22)
    {
        updateRate[10, 0] = updateRate[10, 0] + updates;
    }
    else if (zonaActual == 01)
    {
        updateRate[12, 0] = updateRate[12, 0] + updates;
    }
    else if (zonaActual == 02)
    {
        updateRate[13, 0] = updateRate[13, 0] + updates;
    }
    else if (zonaActual == 0)
    {
        updateRate[14, 0] = updateRate[14, 0] + updates;
    }
    updatesUnitarios[i, 0] = updatesUnitarios[i, 0] + updates;
```

```
                zonaActual = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
                indiceInicial = n;
            }
        }
    }

    //Expected
    tiempoSinSeñalSuperado = false;
    for (int i = 0; i < aircraftDetected.Count(); i++)
    {
        int zonaActual = myList.GetPlanI(aircraftDetected[i][0]).GetZona();
        double tiempoInicial =
myList.GetPlanI(aircraftDetected[i][0]).GetUTCcorregido() * 60;
        for (int n = 1; n < aircraftDetected[i].Count(); n++)
        {
            if (myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() >
myList.GetPlanI(aircraftDetected[i][n - 1]).GetUTCcorregido() + 1)//Si tiempo
actual es más grande que el anterior +1min, dividimos en 2 el vuelo
            {
                tiempoSinSeñalSuperado = true;
            }

            if ((myList.GetPlanI(aircraftDetected[i][n]).GetZona() != zonaActual)
|| (n == aircraftDetected[i].Count() - 1) || (tiempoSinSeñalSuperado))
            {
                tiempoSinSeñalSuperado = false;
                double tiempoFinal;
                if (n != aircraftDetected[i].Count() - 1)//Miramos si se trata del
ultimo mensaje del avion
                    tiempoFinal = myList.GetPlanI(aircraftDetected[i][n -
1]).GetUTCcorregido() * 60;
                else tiempoFinal =
myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() * 60;

                double updates = tiempoFinal - tiempoInicial + 1;
                if (zonaActual == 11)
                {
                    updateRate[1, 1] = updateRate[1, 1] + updates;
                }
                else if (zonaActual == 12)
                {
                    updateRate[2, 1] = updateRate[2, 1] + updates;
                }
                else if (zonaActual == 13)
                {
                    updateRate[3, 1] = updateRate[3, 1] + updates;
                }
                else if (zonaActual == 4)
                {
                    updateRate[4, 1] = updateRate[4, 1] + updates;
```

```
                }
                else if (zonaActual == 31)
                {
                    updateRate[6, 1] = updateRate[6, 1] + updates;
                }
                else if (zonaActual == 32)
                {
                    updateRate[7, 1] = updateRate[7, 1] + updates;
                }
                else if (zonaActual == 21)
                {
                    updateRate[9, 1] = updateRate[9, 1] + updates;
                }
                else if (zonaActual == 22)
                {
                    updateRate[10, 1] = updateRate[10, 1] + updates;
                }
                else if (zonaActual == 01)
                {
                    updateRate[12, 1] = updateRate[12, 1] + updates;
                }
                else if (zonaActual == 02)
                {
                    updateRate[13, 1] = updateRate[13, 1] + updates;
                }
                else if (zonaActual == 0)
                {
                    updateRate[14, 1] = updateRate[14, 1] + updates;
                }
                updatesUnitarios[i, 1] = updatesUnitarios[i, 1] + updates;
                zonaActual = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
                tiempoInicial =
myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() * 60;
            }
        }
    }

    //Totales
    for (int i = 0; i < 2; i++)
    {
        updateRate[0, i] = updateRate[1, i] + updateRate[2, i] + updateRate[3,
i] + updateRate[4, i];//Total maneouvering
        updateRate[5, i] = updateRate[6, i] + updateRate[7, i];//Total Apron
        updateRate[8, i] = updateRate[9, i] + updateRate[10, i];//Total Stand
        updateRate[11, i] = updateRate[12, i] + updateRate[13, i] +
updateRate[14, i];//Airborne
    }

    //Update rates
    for (int i = 0; i < updateRate.GetLength(0); i++)
```

```
        {
            if (updateRate[i, 1] != 0)
                updateRate[i, 2] = updateRate[i, 0]*100 / updateRate[i, 1];//%
            else updateRate[i, 2] = 0;
        }

        // Update rate minimo segun doc ED117
        updateRate[0, 3] = 95;//%
        updateRate[5, 3] = 70;
        updateRate[8, 3] = 50;
        updateRate[11, 3] = 95;
    }

    public void SetPositionAccuracy()
    {
        PosAcc = new double[12, 7];
        List<double[]> diferencia = new List<double[]>(2);//posición 1=
diferencia; posicion 2=zona

        for (int i = 0; i < diferencias.Count(); i++)
        {
            double[] diferenciaActual = new double[2]; //[0]=Sqrt((X1-X2)^2+(Y1-
Y2)^2); [1]=Zona
            diferenciaActual[0] = Math.Sqrt(Math.Pow(diferencias[i][0], 2) +
Math.Pow(diferencias[i][1], 2));
            diferenciaActual[1] = diferencias[i][2];
            diferencia.Add(diferenciaActual);
        }

        List<int> zonas = new List<int>();

        for (int i = 0; i < PosAcc.GetLength(0); i++)//Inicialmente -1. Si no hay
información de la zona se quitará el -1 por null
        {
            for (int j = 0; j < PosAcc.GetLength(1); j++)
            {
                if ((j != 3) && (j != 4))
                {
                    PosAcc[i, j] = -1;
                }
            }
        }

        //P95, P99, Mean
        zonas.Add(11);zonas.Add(12); zonas.Add(13); zonas.Add(4); //Total
Maneouvering
        SetPosAccValues(0, diferencia, zonas);
        zonas.Clear();

        zonas.Add(11);  //11
```

```
SetPosAccValues(1, diferencia, zonas);
zonas.Clear();

zonas.Add(12);  //12
SetPosAccValues(2, diferencia, zonas);
zonas.Clear();

zonas.Add(13);
SetPosAccValues(3, diferencia, zonas);
zonas.Clear();

zonas.Add(4); //4
SetPosAccValues(4, diferencia, zonas);
zonas.Clear();

zonas.Add(31); zonas.Add(32); //Total Apron
SetPosAccValues(5, diferencia, zonas);
zonas.Clear();

zonas.Add(31);  //31
SetPosAccValues(6, diferencia, zonas);
zonas.Clear();

zonas.Add(32); //32
SetPosAccValues(7, diferencia, zonas);
zonas.Clear();

zonas.Add(21); zonas.Add(22); //Total Stand
SetPosAccValues(8, diferencia, zonas);
zonas.Clear();

zonas.Add(21); //21
SetPosAccValues(9, diferencia, zonas);
zonas.Clear();

zonas.Add(22); //22
SetPosAccValues(10, diferencia, zonas);
zonas.Clear();

//Max Value Detected
double promedioTemporal = 0;
double zonaActual = diferencia[0][1];
int recuento = 1; //mira si la ventana llega a 5
for (int i = 1; i < diferencia.Count(); i++)
{
   if (diferencia[i][1] == zonaActual)
   {
      recuento++;
   }
   else
```

```
            {
                recuento = 1;
                zonaActual = diferencia[i][1];
            }

            if (recuento == 5)//Ventana de 5
            {
                promedioTemporal = (diferencia[i][1] + diferencia[i - 1][1] +
diferencia[i - 2][1] + diferencia[i - 3][1] + diferencia[i - 4][1]) / 5;
                if (zonaActual == 11)
                {
                    if (promedioTemporal> PosAcc[1, 2])
                        PosAcc[1, 2] = promedioTemporal;
                }
                else if (zonaActual == 12)
                {
                    if (promedioTemporal > PosAcc[2, 2])
                        PosAcc[2, 2] = promedioTemporal;
                }
                else if (zonaActual == 13)
                {
                    if (promedioTemporal > PosAcc[3, 2])
                        PosAcc[3, 2] = promedioTemporal;
                }
                else if (zonaActual == 4)
                {
                    if (promedioTemporal > PosAcc[4, 2])
                        PosAcc[4, 2] = promedioTemporal;
                }
                else if (zonaActual == 31)
                {
                    if (promedioTemporal > PosAcc[6, 2])
                        PosAcc[6, 2] = promedioTemporal;
                }
                else if (zonaActual == 32)
                {
                    if (promedioTemporal > PosAcc[7, 2])
                        PosAcc[7, 2] = promedioTemporal;
                }
                else if (zonaActual == 21)
                {
                    if (promedioTemporal > PosAcc[9, 2])
                        PosAcc[9, 2] = promedioTemporal;
                }
                else if (zonaActual == 22)
                {
                    if (promedioTemporal > PosAcc[10, 2])
                        PosAcc[10, 2] = promedioTemporal;
                }
```

```
                promedioTemporal = 0;
                recuento = 1;
                i++;
                if (i < diferencia.Count())
                {
                    zonaActual = diferencia[i][1];
                }
            }
        }

        double promedioSeccion = PosAcc[1, 2];//Promedio Seccion
Maneouvering Area
        for (int i = 2; i <= 4; i++)
        {
            if (PosAcc[i, 2] > promedioSeccion)
            {
                promedioSeccion = PosAcc[i, 2];
            }
        }
        PosAcc[0, 2] = promedioSeccion;

        if (PosAcc[7, 2] > PosAcc[6, 2]) //Promedio Seccion Apron
        {
            PosAcc[5, 2] = PosAcc[7, 2];
        }
        else PosAcc[5, 2] = PosAcc[6, 2];

        if (PosAcc[10, 2] > PosAcc[9, 2]) //Promedio Seccion Stand
        {
            PosAcc[8, 2] = PosAcc[10, 2];
        }
        else PosAcc[8, 2] = PosAcc[9, 2];

        PosAcc[0, 3] = 7.5; //Max Maneouvering P95
        PosAcc[0, 4] = 12; //Max Maneouvering P99
        PosAcc[5, 3] = 7.5; //Max Apron P95
        PosAcc[5, 4] = 12; //Max Apron P99
    }

    public void SetProbOfMLATDetection()
    {
        MLATDet = new double[11, 4];

        //Detecciones reales & Expected
        bool tiempoSinSeñalSuperado = false; //Si tiempo actual es más grande
que el anterior +1min
        for (int i = 0; i < aircraftDetected.Count(); i++)
        {
            int zonaActual = myList.GetPlanI(aircraftDetected[i][0]).GetZona();
            int indiceInicial = 0;
```

```
            double tiempoInicial =
myList.GetPlanI(aircraftDetected[i][0]).GetUTCcorregido() * 60;
            for (int n = 1; n < aircraftDetected[i].Count(); n++)
            {
                if (myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() >
myList.GetPlanI(aircraftDetected[i][n - 1]).GetUTCcorregido() + 1)//Si tiempo
actual es más grande que el anterior +1min, dividimos en 2 el vuelo
                {
                    tiempoSinSeñalSuperado = true;
                }

                if ((myList.GetPlanI(aircraftDetected[i][n]).GetZona() != zonaActual)
|| (n == aircraftDetected[i].Count() - 1) || (tiempoSinSeñalSuperado))
                {
                    tiempoSinSeñalSuperado = false;
                    if ((zonaActual == 11) || (zonaActual == 12) || (zonaActual == 13)
|| (zonaActual == 4) || (zonaActual == 21) || (zonaActual == 22) || (zonaActual
== 31)|| (zonaActual == 32))
                    {
                        int indiceFinal;
                        if (n != aircraftDetected[i].Count() - 1)
                            indiceFinal = n - 1;
                        else indiceFinal = n;

                        double ventanaTiempo = 2;//En segundos
                        if ((zonaActual == 21) || (zonaActual == 22))
                        {
                            ventanaTiempo = 5;
                        }

                        double tiempoFinal =
myList.GetPlanI(aircraftDetected[i][indiceFinal]).GetUTCcorregido() * 60;
                        double tiempoFinalVentana = tiempoInicial + ventanaTiempo;
                        int ventanas = 0;
                        int ventanasDetectadas = 0;

                        while (tiempoFinalVentana <= tiempoFinal)
                        {
                            bool ventanaCumple = false;
                            int m = indiceInicial;
                            while ((m <= indiceFinal) && (!ventanaCumple))
                            {
                                if
(myList.GetPlanI(aircraftDetected[i][m]).GetUTCcorregido() * 60 >= tiempoInicial
&& myList.GetPlanI(aircraftDetected[i][m]).GetUTCcorregido() * 60 <=
tiempoFinalVentana)
                                {
                                    ventanaCumple = true;
                                    ventanasDetectadas++;
                                }
```

```
                m++;
            }
            ventanas++;
            tiempoInicial = tiempoInicial + 1;
            tiempoFinalVentana = tiempoInicial + ventanaTiempo;
        }

        if (zonaActual == 11)
        {
            MLATDet[1, 0] = MLATDet[1, 0] + ventanasDetectadas;
            MLATDet[1, 1] = MLATDet[1, 1] + ventanas;
        }
        else if (zonaActual == 12)
        {
            MLATDet[2, 0] = MLATDet[2, 0] + ventanasDetectadas;
            MLATDet[2, 1] = MLATDet[2, 1] + ventanas;
        }
        else if (zonaActual == 13)
        {
            MLATDet[3, 0] = MLATDet[3, 0] + ventanasDetectadas;
            MLATDet[3, 1] = MLATDet[3, 1] + ventanas;
        }
        else if (zonaActual == 4)
        {
            MLATDet[4, 0] = MLATDet[4, 0] + ventanasDetectadas;
            MLATDet[4, 1] = MLATDet[4, 1] + ventanas;
        }
        else if (zonaActual == 31)
        {
            MLATDet[5, 0] = MLATDet[5, 0] + ventanasDetectadas;
            MLATDet[5, 1] = MLATDet[5, 1] + ventanas;
        }
        else if (zonaActual == 32)
        {
            MLATDet[6, 0] = MLATDet[6, 0] + ventanasDetectadas;
            MLATDet[6, 1] = MLATDet[6, 1] + ventanas;
        }
        else if (zonaActual == 21)
        {
            MLATDet[8, 0] = MLATDet[8, 0] + ventanasDetectadas;
            MLATDet[8, 1] = MLATDet[8, 1] + ventanas;
        }
        else if (zonaActual == 22)
        {
            MLATDet[9, 0] = MLATDet[9, 0] + ventanasDetectadas;
            MLATDet[9, 1] = MLATDet[9, 1] + ventanas;
        }
        //Airborne not required
    }
    zonaActual = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
```

```
                    indiceInicial = n;
                    tiempoInicial =
myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() * 60;
                }
            }
        }

        //Totales
        for (int i = 0; i < 2; i++)
        {
            MLATDet[0, i] = MLATDet[1, i] + MLATDet[2, i] + MLATDet[3, i] +
MLATDet[4, i] + MLATDet[5, i] + MLATDet[6, i]; //Total maneouvering + Apron
            MLATDet[7, i] = MLATDet[8, i] + MLATDet[9, i]; //Total Stand
        }

        //Prob. of detection
        for (int i = 0; i < MLATDet.GetLength(0); i++)
        {
            if (MLATDet[i, 1] != 0)
                MLATDet[i, 2] = MLATDet[i, 0] * 100 / MLATDet[i, 1];
            else MLATDet[i, 2] = 0;
        }

        //Prob. of detection minimo segun doc ED117
        MLATDet[0, 3] = 99; //[%]
        MLATDet[7, 3] = 99;
    }

    public void SetProbOfIdentification()
    {
        Identification = new double[16, 4];

        //Incorrect & Correct
        List<string> ICAOdiferentes = new List<string>();
        List<int> ICAOdiferentesCount = new List<int>();
        for (int i = 0; i < aircraftTrackDetected.Count(); i++)
        {
            ICAOdiferentes.Clear();
            ICAOdiferentesCount.Clear();

ICAOdiferentes.Add(myList.GetPlanI(aircraftTrackDetected[i][0]).GetICAOAdres
s());
            ICAOdiferentesCount.Add(1);
            for (int n = 1; n < aircraftTrackDetected[i].Count(); n++)
            {
                for (int m = 0; m < ICAOdiferentes.Count(); m++)
                {
                    if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetICAOAdress()
== ICAOdiferentes[m])
                    {
```

```
                    List<int> ICAOdiferentesCountTemporal = new List<int>();
                    for (int j = 0; j < ICAOdiferentesCount.Count(); j++)
                    {
                        if (m == j)
                        {

ICAOdiferentesCountTemporal.Add(ICAOdiferentesCount[j] + 1);
                        }
                        else
ICAOdiferentesCountTemporal.Add(ICAOdiferentesCount[j]);
                    }
                    ICAOdiferentesCount.Clear();
                    ICAOdiferentesCount = ICAOdiferentesCountTemporal;
                }
                else
                {

ICAOdiferentes.Add(myList.GetPlanI(aircraftTrackDetected[i][n]).GetICAOAdres
s());
                    ICAOdiferentesCount.Add(1);
                }
            }
        }
        int posicionGanadora = 0;
        if (ICAOdiferentesCount.Count() > 1)
        {
            for (int m = 1; m < ICAOdiferentesCount.Count(); m++)
            {
                if (ICAOdiferentesCount[m] >
ICAOdiferentesCount[posicionGanadora])
                {
                    posicionGanadora = m;
                }
            }
        }

        for (int n = 0; n < aircraftTrackDetected[i].Count(); n++)
        {
            int correcto = 0;
            int incorrecto = 0;
            if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetICAOAdress() !=
ICAOdiferentes[posicionGanadora])
            {
                incorrecto = 1;
            }
            else correcto = 1;

            if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() == 11)
            {
                Identification[1, 0] = Identification[1, 0] + correcto;
```

```
                Identification[1, 1] = Identification[1, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
12)
            {
                Identification[2, 0] = Identification[2, 0] + correcto;
                Identification[2, 1] = Identification[2, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
13)
            {
                Identification[3, 0] = Identification[3, 0] + correcto;
                Identification[3, 1] = Identification[3, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() == 4)
            {
                Identification[4, 0] = Identification[4, 0] + correcto;
                Identification[4, 1] = Identification[4, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
31)
            {
                Identification[6, 0] = Identification[6, 0] + correcto;
                Identification[6, 1] = Identification[6, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
32)
            {
                Identification[7, 0] = Identification[7, 0] + correcto;
                Identification[7, 1] = Identification[7, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
21)
            {
                Identification[9, 0] = Identification[9, 0] + correcto;
                Identification[9, 1] = Identification[9, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
22)
            {
                Identification[10, 0] = Identification[10, 0] + correcto;
                Identification[10, 1] = Identification[10, 1] + incorrecto;
            }
            else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
01)
            {
                Identification[12, 0] = Identification[12, 0] + correcto;
                Identification[12, 1] = Identification[12, 1] + incorrecto;
            }
```

```
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
02)
                {
                    Identification[13, 0] = Identification[13, 0] + correcto;
                    Identification[13, 1] = Identification[13, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() == 0)
                {
                    Identification[14, 0] = Identification[14, 0] + correcto;
                    Identification[14, 1] = Identification[14, 1] + incorrecto;
                }
            }
        }

        //Totales
        for (int i = 0; i < 2; i++)
        {
            Identification[0, i] = Identification[1, i] + Identification[2, i] +
Identification[3, i] + Identification[4, i];//Total maneouvering
            Identification[5, i] = Identification[6, i] + Identification[7, i];//Total Apron
            Identification[8, i] = Identification[9, i] + Identification[10, i];//Total
Stand
            Identification[11, i] = Identification[12, i] + Identification[13, i] +
Identification[14, i];//Airborne
            Identification[15, i] = Identification[0, i] + Identification[5, i] +
Identification[8, i] + Identification[11, i]; //Suma de todos
        }

        //PID
        for (int i = 0; i < Identification.GetLength(0); i++)
        {
            if (Identification[i, 0] + Identification[i, 1] != 0)
            {
                Identification[i, 2] = Identification[i, 0] * 100 / (Identification[i, 0] +
Identification[i, 1]);
            }
        }

        // PID minimo segun doc ED117
        Identification[15, 3] = 99; //[%]
    }

    public void SetProbOfFalseDetection()
    {
        FalseDetection = new double[15, 4];

        //Reports
        for (int i = 0; i < aircraftDetected.Count(); i++)
        {
            for (int n = 0; n < aircraftDetected[i].Count(); n++)
```

```
        {
            int zonaActual = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
            if (zonaActual == 11)
            {
                FalseDetection[1, 0] = FalseDetection[1, 0] + 1;
            }
            else if (zonaActual == 12)
            {
                FalseDetection[2, 0] = FalseDetection[2, 0] + 1;
            }
            else if (zonaActual == 13)
            {
                FalseDetection[3, 0] = FalseDetection[3, 0] + 1;
            }
            else if (zonaActual == 4)
            {
                FalseDetection[4, 0] = FalseDetection[4, 0] + 1;
            }
            else if (zonaActual == 31)
            {
                FalseDetection[6, 0] = FalseDetection[6, 0] + 1;
            }
            else if (zonaActual == 32)
            {
                FalseDetection[7, 0] = FalseDetection[7, 0] + 1;
            }
            else if (zonaActual == 21)
            {
                FalseDetection[9, 0] = FalseDetection[9, 0] + 1;
            }
            else if (zonaActual == 22)
            {
                FalseDetection[10, 0] = FalseDetection[10, 0] + 1;
            }
            else if (zonaActual == 01)
            {
                FalseDetection[12, 0] = FalseDetection[12, 0] + 1;
            }
            else if (zonaActual == 02)
            {
                FalseDetection[13, 0] = FalseDetection[13, 0] + 1;
            }
        }
    }

    //False Reports
    for (int i = 0; i < aircraftDetected.Count(); i++)
    {
        int zonaInicial = myList.GetPlanI(aircraftDetected[i][0]).GetZona();
```

```
            double tiempoInicial =
myList.GetPlanI(aircraftDetected[i][0]).GetUTCcorregido() * 60;
            double[] posicionInicial = GetPosicion(aircraftDetected[i][0]);
            for (int n = 1; n < aircraftDetected[i].Count(); n++)
            {
                int zonaActual = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
                double tiempoActual =
myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() * 60;
                double[] posicionActual = GetPosicion(aircraftDetected[i][n]);
                bool falseDetection = false;//Controla si ha habido false detection
                if ((zonaInicial != 0) && (zonaInicial != -1))
                {
                    if ((zonaActual != 0) && (zonaActual != -1))
                    {
                        bool mismaZona = false; //solo podemos comparar zonas
iguales, ya que las distancia umbral cambia
                        int distUmbral = 50; //[m]. Para la zona aeroportuaria
                        if (zonaInicial == 01)
                        {
                            if (zonaActual == 01)
                            {
                                mismaZona = true;
                                distUmbral = 80; //[m]
                            }
                        }
                        else if (zonaInicial == 02)
                        {
                            if (zonaActual == 02)
                            {
                                mismaZona = true;
                                distUmbral = 160; //[m]
                            }
                        }
                        else//No es 0, -1, 01, 02; por lo que distUmbral será 50
                        {
                            if ((zonaActual != 01) && (zonaActual != 02))
                            {
                                mismaZona = true;
                            }
                        }

                        if (mismaZona)//Solo comparamos zonas iguales: zona
aeroportuaria, zona 4 o zona 5
                        {
                            if (tiempoActual <= tiempoInicial + 1)//Si la señal n llega 1s
despues de la señal n-1
                            {
                                if
((myList.GetPlanI(aircraftDetected[i][n]).GetCartTrackVelX() != Math.Pow(10,
```

```
8)) && (myList.GetPlanI(aircraftDetected[i][n]).GetCartTrackVelY() !=
Math.Pow(10, 8)))
                        {
                            posicionInicial[0] = posicionInicial[0]+
myList.GetPlanI(aircraftDetected[i][n]).GetCartTrackVelX(); //X=Xo+V*t (t=1s)
==> X=Xo+V
                            posicionInicial[1] = posicionInicial[1] +
myList.GetPlanI(aircraftDetected[i][n]).GetCartTrackVelY(); //Calculamos punto
ficticio
                            double distancia =
Math.Sqrt(Math.Pow(posicionActual[0] - posicionInicial[0], 2) +
Math.Pow(posicionActual[1] - posicionInicial[1], 2));
                            if (distancia > distUmbral)
                            {
                                falseDetection = true;
                                n++;//Saltamos un mensaje, ya que el mensaje
falso seguramente esté a >50m del anterior y también del siguiente
                            }
                        }
                    }
                }
                if (falseDetection)
                {
                    if (zonaActual == 11)
                    {
                        FalseDetection[1, 1] = FalseDetection[1, 1] + 1;
                    }
                    else if (zonaActual == 12)
                    {
                        FalseDetection[2, 1] = FalseDetection[2, 1] + 1;
                    }
                    else if (zonaActual == 13)
                    {
                        FalseDetection[3, 1] = FalseDetection[3, 1] + 1;
                    }
                    else if (zonaActual == 4)
                    {
                        FalseDetection[4, 1] = FalseDetection[4, 1] + 1;
                    }
                    else if (zonaActual == 31)
                    {
                        FalseDetection[6, 1] = FalseDetection[6, 1] + 1;
                    }
                    else if (zonaActual == 32)
                    {
                        FalseDetection[7, 1] = FalseDetection[7, 1] + 1;
                    }
                    else if (zonaActual == 21)
                    {
                        FalseDetection[9, 1] = FalseDetection[9, 1] + 1;
```

```
                        }
                        else if (zonaActual == 22)
                        {
                            FalseDetection[10, 1] = FalseDetection[10, 1] + 1;
                        }
                        else if (zonaActual == 01)
                        {
                            FalseDetection[12, 1] = FalseDetection[12, 1] + 1;
                        }
                        else if (zonaActual == 02)
                        {
                            FalseDetection[13, 1] = FalseDetection[13, 1] + 1;
                        }
                    }
                }
            }
            if (falseDetection)
            {
                if (n < aircraftDetected[i].Count() - 1)//Para que no salte error con
la ultima posicion del vector
                {
                    zonaInicial = myList.GetPlanI(aircraftDetected[i][n]).GetZona();
                    tiempoInicial =
myList.GetPlanI(aircraftDetected[i][n]).GetUTCcorregido() * 60;
                    posicionInicial = GetPosicion(aircraftDetected[i][n]);
                }
                //if not, el bucle acabará en este ciclo
            }
            else
            {
                zonaInicial = zonaActual;
                tiempoInicial = tiempoActual;
                posicionInicial = posicionActual;
            }
        }
    }

    //Totales
    for (int i = 0; i < 2; i++)
    {
        FalseDetection[0, i] = FalseDetection[1, i] + FalseDetection[2, i] +
FalseDetection[3, i] + FalseDetection[4, i];//Total maneouvering
        FalseDetection[5, i] = FalseDetection[6, i] + FalseDetection[7,
i];//Total Apron
        FalseDetection[8, i] = FalseDetection[9, i] + FalseDetection[10,
i];//Total Stand
        FalseDetection[11, i] = FalseDetection[12, i] + FalseDetection[13,
i];//Airborne (solo tipo 4 y 5)
        FalseDetection[14, i] = FalseDetection[0, i] + FalseDetection[5, i] +
FalseDetection[8, i] + FalseDetection[11, i]; //Suma de todos
```

```
        }

        //Prob. of detection
        for (int i = 0; i < FalseDetection.GetLength(0); i++)
        {
            if (FalseDetection[i, 0] != 0)
                FalseDetection[i, 2] = FalseDetection[i, 1] * 100 / FalseDetection[i,
0]; // [%]
            else FalseDetection[i, 2] = 0;
        }

        //Prob. of detection minimo segun doc ED117
        FalseDetection[14, 3] = 0.01; // [%]
    }

    public void SetProbOfFalseDetectionDGPS()
    {
        FalseDetection = new double[15, 4];
        List<double[]> diferencia = new List<double[]>(2);//posición 1=
diferencia; posicion 2=zona

        for (int i = 0; i < DGPSList.GetNumList(); i++)
        {
            if (DGPSList.GetPlanI(i).GetIndiceDGPS() != -1)
            {
                double[] diferenciaActual = new double[2]; //[0]=Sqrt((X1-
X2)^2+(Y1-Y2)^2); [1]=Zona
                diferenciaActual[0] =
Math.Sqrt(Math.Pow(DGPSList.GetPlanI(i).GetPosicion()[0] -
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetPosicion()[0], 2) +
Math.Pow(DGPSList.GetPlanI(i).GetPosicion()[1] -
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetPosicion()[1], 2));
                diferenciaActual[1] =
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetZona();
                diferencia.Add(diferenciaActual);
            }
        }

        //Reports & False Reports
        for (int i = 0; i < diferencia.Count(); i++)
        {
            double zonaActual = diferencia[i][1];
            int falso = 0;
            if (diferencia[i][0] > 50)
            {
                falso = 1;
            }
            if (zonaActual == 11)
            {
                FalseDetection[1, 0] = FalseDetection[1, 0] + 1;
```

```
            FalseDetection[1, 1] = FalseDetection[1, 1] + falso;
        }
        else if (zonaActual == 12)
        {
            FalseDetection[2, 0] = FalseDetection[2, 0] + 1;
            FalseDetection[2, 1] = FalseDetection[2, 1] + falso;
        }
        else if (zonaActual == 13)
        {
            FalseDetection[3, 0] = FalseDetection[3, 0] + 1;
            FalseDetection[3, 1] = FalseDetection[3, 1] + falso;
        }
        else if (zonaActual == 4)
        {
            FalseDetection[4, 0] = FalseDetection[4, 0] + 1;
            FalseDetection[4, 1] = FalseDetection[4, 1] + falso;
        }
        else if (zonaActual == 31)
        {
            FalseDetection[6, 0] = FalseDetection[6, 0] + 1;
            FalseDetection[6, 1] = FalseDetection[6, 1] + falso;
        }
        else if (zonaActual == 32)
        {
            FalseDetection[7, 0] = FalseDetection[7, 0] + 1;
            FalseDetection[7, 1] = FalseDetection[7, 1] + falso;
        }
        else if (zonaActual == 21)
        {
            FalseDetection[9, 0] = FalseDetection[9, 0] + 1;
            FalseDetection[9, 1] = FalseDetection[9, 1] + falso;
        }
        else if (zonaActual == 22)
        {
            FalseDetection[10, 0] = FalseDetection[10, 0] + 1;
            FalseDetection[10, 1] = FalseDetection[10, 1] + falso;
        }
        else if (zonaActual == 01)
        {
            FalseDetection[12, 0] = FalseDetection[12, 0] + 1;
            FalseDetection[12, 1] = FalseDetection[12, 1] + falso;
        }
        else if (zonaActual == 02)
        {
            FalseDetection[13, 0] = FalseDetection[13, 0] + 1;
            FalseDetection[13, 1] = FalseDetection[13, 1] + falso;
        }
    }

    //Totales
```

```
            for (int i = 0; i < 2; i++)
            {
                FalseDetection[0, i] = FalseDetection[1, i] + FalseDetection[2, i] +
FalseDetection[3, i] + FalseDetection[4, i];//Total maneouvering
                FalseDetection[5, i] = FalseDetection[6, i] + FalseDetection[7,
i];//Total Apron
                FalseDetection[8, i] = FalseDetection[9, i] + FalseDetection[10,
i];//Total Stand
                FalseDetection[11, i] = FalseDetection[12, i] + FalseDetection[13,
i];//Airborne (solo tipo 4 y 5)
                FalseDetection[14, i] = FalseDetection[0, i] + FalseDetection[5, i] +
FalseDetection[8, i] + FalseDetection[11, i]; //Suma de todos
            }

            //Prob. of detection
            for (int i = 0; i < FalseDetection.GetLength(0); i++)
            {
                if (FalseDetection[i, 0] != 0)
                    FalseDetection[i, 2] = FalseDetection[i, 1] * 100 / FalseDetection[i,
0]; // [%]
                else FalseDetection[i, 2] = 0;
            }

            //Prob. of detection minimo segun doc ED117
            FalseDetection[14, 3] = 0.01; // [%]
        }

        public void SetProbOfFalseIdentification()
        {
            FalseIdentification = new double[16, 4];

            //Incorrect & Correct
            List<string> ICAOdiferentes = new List<string>();
            List<int> ICAOdiferentesCount = new List<int>();
            for (int i = 0; i < aircraftTrackDetected.Count(); i++)
            {
                ICAOdiferentes.Clear();
                ICAOdiferentesCount.Clear();

ICAOdiferentes.Add(myList.GetPlanI(aircraftTrackDetected[i][0]).GetICAOAdres
s());
                ICAOdiferentesCount.Add(1);
                for (int n = 1; n < aircraftTrackDetected[i].Count(); n++)
                {
                    for (int m = 0; m < ICAOdiferentes.Count(); m++)
                    {
                        if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetICAOAdress()
== ICAOdiferentes[m])
                        {
                            List<int> ICAOdiferentesCountTemporal = new List<int>();
```

```
                    for (int j = 0; j < ICAOdiferentesCount.Count(); j++)
                    {
                        if (m == j)
                        {

ICAOdiferentesCountTemporal.Add(ICAOdiferentesCount[j] + 1);
                        }
                        else
ICAOdiferentesCountTemporal.Add(ICAOdiferentesCount[j]);
                    }
                    ICAOdiferentesCount.Clear();
                    ICAOdiferentesCount = ICAOdiferentesCountTemporal;
                }
                else
                {

ICAOdiferentes.Add(myList.GetPlanI(aircraftTrackDetected[i][n]).GetICAOAdres
s());
                    ICAOdiferentesCount.Add(1);
                }
            }
        }
        int posicionGanadora = 0;
        if (ICAOdiferentesCount.Count() > 1)
        {
            for (int m = 1; m < ICAOdiferentesCount.Count(); m++)
            {
                if (ICAOdiferentesCount[m] >
ICAOdiferentesCount[posicionGanadora])
                {
                    posicionGanadora = m;
                }
            }
        }

        int zonaActual =
myList.GetPlanI(aircraftTrackDetected[i][0]).GetZona();
        int indiceInicial = 0;
        double tiempoInicial =
myList.GetPlanI(aircraftTrackDetected[i][0]).GetUTCcorregido() * 60;
        for (int n = 1; n < aircraftTrackDetected[i].Count(); n++)
        {
            if ((myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() !=
zonaActual) || (n == aircraftTrackDetected[i].Count() - 1))
            {
                int indiceFinal;
                if (n != aircraftTrackDetected[i].Count() - 1)
                    indiceFinal = n - 1;
                else indiceFinal = n;
```

```
                double ventanaTiempo = 5;//En segundos

                double tiempoFinal =
myList.GetPlanI(aircraftTrackDetected[i][indiceFinal]).GetUTCcorregido() * 60;
                double tiempoFinalVentana = tiempoInicial + ventanaTiempo;
                int ventanas = 0;
                int incorrecto = 0;

                while (tiempoFinalVentana <= tiempoFinal)
                {
                    bool ventanaCumple = true;
                    int m = indiceInicial;
                    while ((m <= indiceFinal) && (ventanaCumple))
                    {
                        if
(myList.GetPlanI(aircraftTrackDetected[i][m]).GetUTCcorregido() * 60 >=
tiempoInicial &&
myList.GetPlanI(aircraftTrackDetected[i][m]).GetUTCcorregido() * 60 <=
tiempoFinalVentana)
                        {

if(myList.GetPlanI(aircraftTrackDetected[i][m]).GetICAOAdress()!=
ICAOdiferentes[posicionGanadora])
                            {
                                ventanaCumple = false;
                                incorrecto++;
                            }
                        }
                        m++;
                    }
                    ventanas++;
                    tiempoInicial = tiempoInicial + 1;
                    tiempoFinalVentana = tiempoInicial + ventanaTiempo;
                }

                if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() == 11)
                {
                    FalseIdentification[1, 0] = FalseIdentification[1, 0] + ventanas;
                    FalseIdentification[1, 1] = FalseIdentification[1, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
12)
                {
                    FalseIdentification[2, 0] = FalseIdentification[2, 0] + ventanas;
                    FalseIdentification[2, 1] = FalseIdentification[2, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
13)
                {
                    FalseIdentification[3, 0] = FalseIdentification[3, 0] + ventanas;
```

```
                    FalseIdentification[3, 1] = FalseIdentification[3, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
4)
                {
                    FalseIdentification[4, 0] = FalseIdentification[4, 0] + ventanas;
                    FalseIdentification[4, 1] = FalseIdentification[4, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
31)
                {
                    FalseIdentification[6, 0] = FalseIdentification[6, 0] + ventanas;
                    FalseIdentification[6, 1] = FalseIdentification[6, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
32)
                {
                    FalseIdentification[7, 0] = FalseIdentification[7, 0] + ventanas;
                    FalseIdentification[7, 1] = FalseIdentification[7, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
21)
                {
                    FalseIdentification[9, 0] = FalseIdentification[9, 0] + ventanas;
                    FalseIdentification[9, 1] = FalseIdentification[9, 1] + incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
22)
                {
                    FalseIdentification[10, 0] = FalseIdentification[10, 0] +
ventanas;
                    FalseIdentification[10, 1] = FalseIdentification[10, 1] +
incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
01)
                {
                    FalseIdentification[12, 0] = FalseIdentification[12, 0] +
ventanas;
                    FalseIdentification[12, 1] = FalseIdentification[12, 1] +
incorrecto;
                }
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
02)
                {
                    FalseIdentification[13, 0] = FalseIdentification[13, 0] +
ventanas;
                    FalseIdentification[13, 1] = FalseIdentification[13, 1] +
incorrecto;
                }
```

```
                else if (myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona() ==
0)
                {
                    FalseIdentification[14, 0] = FalseIdentification[14, 0] +
ventanas;
                    FalseIdentification[14, 1] = FalseIdentification[14, 1] +
incorrecto;
                }
                zonaActual =
myList.GetPlanI(aircraftTrackDetected[i][n]).GetZona();
                indiceInicial = n;
                tiempoInicial =
myList.GetPlanI(aircraftTrackDetected[i][n]).GetUTCcorregido() * 60;
            }
        }
    }

    //Totales
    for (int i = 0; i < 2; i++)
    {
        FalseIdentification[0, i] = FalseIdentification[1, i] +
FalseIdentification[2, i] + FalseIdentification[3, i] + FalseIdentification[4, i];//Total
maneouvering
        FalseIdentification[5, i] = FalseIdentification[6, i] +
FalseIdentification[7, i];//Total Apron
        FalseIdentification[8, i] = FalseIdentification[9, i] +
FalseIdentification[10, i];//Total Stand
        FalseIdentification[11, i] = FalseIdentification[12, i] +
FalseIdentification[13, i] + FalseIdentification[14, i];//Airborne
        FalseIdentification[15, i] = FalseIdentification[0, i] +
FalseIdentification[5, i] + FalseIdentification[8, i] + FalseIdentification[11, i];
//Suma de todos
    }

    //PFI
    for (int i = 0; i < FalseIdentification.GetLength(0); i++)
    {
        if (FalseIdentification[i, 0] != 0)
            FalseIdentification[i, 2] = FalseIdentification[i, 1] * 100 /
FalseIdentification[i, 0];
        else FalseIdentification[i, 2] = 0;
    }

    // PID minimo segun doc ED117
    FalseIdentification[15, 3] = 0.0001; //[%]
}

public List<double[]> SetDiferencias()
{
    List<double[]>  diferenciasFinales = new List<double[]>();
```

```
            for (int i = 0; i < DGPSList.GetNumList(); i++)
            {
                if (DGPSList.GetPlanI(i).GetIndiceDGPS() != -1)
                {
                    double[] diferenciaActual = new double[3];
                    diferenciaActual[0] = DGPSList.GetPlanI(i).GetPosicion()[0] -
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetPosicion()[0];
                    diferenciaActual[1] = DGPSList.GetPlanI(i).GetPosicion()[1] -
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetPosicion()[1];
                    diferenciaActual[2] =
listaPA.GetPlanI(DGPSList.GetPlanI(i).GetIndiceDGPS()).GetZona();
                    diferenciasFinales.Add(diferenciaActual);
                }
            }
            return (diferenciasFinales);
        }

        public void OrdenarDiferencias(double[,] lista, string orden) //Te ordena de
menor a mayor ó de mayor a menor (segun variable orden) las diferecias de un
vector, ordena la primera columna, sirve para n columnas
        {
            int i, j, k;
            double[] temp = new double[lista.GetLength(1)];
            bool algoCambiado;//Si en algun momento para de cambiar, paramos el
bucle
            bool parar = false;//para el bucle
            i = 1;
            while ((i < lista.GetLength(0))&&(!parar))//Algoritmo que ordena de
menor a mayor (metodo burbuja, Bubble Sort)
            {
                algoCambiado = false;
                for (j = 0; j < lista.GetLength(0) - 1; j++) //De menor a mayor
                {
                    if (orden == "mM")
                    {
                        if (lista[j, 0] > lista[j + 1, 0])
                        {
                            for (k = 0; k < lista.GetLength(1); k++)
                            {
                                temp[k] = lista[j, k];
                                lista[j, k] = lista[j + 1, k];
                                lista[j + 1, k] = temp[k];
                                algoCambiado = true;
                            }
                        }
                    }
                    else if (orden=="Mm") //De mayor a menor
                    {
                        if (lista[j, 0] < lista[j + 1, 0])
                        {
```

```
                    for (k = 0; k < lista.GetLength(1); k++)
                    {
                        temp[k] = lista[j, k];
                        lista[j, k] = lista[j + 1, k];
                        lista[j + 1, k] = temp[k];
                        algoCambiado = true;
                    }
                }
            }
        }
        if (!algoCambiado)
        {
            parar = true;
        }
        i++;
    }
}

    public List<double> OrdenarDiferenciasPA(List<double[]> lista, List<int>
zonas) //Te ordena de menor a mayor las diferecias de una lista, teniendo en
cuenta las zonas que comparar. Adaptado al formato de entrada de PosAcc
    {
        List<double> listaTemporal = new List<double>();
        int i, j;
        for (i = 0; i < lista.Count(); i++)//Seleccionamos los componentes de lista
que tienen la zona de interes
        {
            for (j = 0; j < zonas.Count(); j++)
            {
                if (lista[i][1] == zonas[j])
                {
                    listaTemporal.Add(lista[i][0]);
                }
            }
        }

        double[] listaTemporalVector = new double[listaTemporal.Count()];
        for (i = 0; i < listaTemporal.Count(); i++)//Pasamos de Lista a vector
para poder trabajar con mas sencillez en el algoritmo que ordena las
diferencias
        {
            listaTemporalVector[i] = listaTemporal[i];
        }

        double temp;
        bool algoCambiado;//Si en algun momento para de cambiar, paramos el
bucle
        bool parar = false;//para el bucle
        i = 1;
        while ((i < listaTemporalVector.GetLength(0))&&(!parar))
```

```
        {
            algoCambiado = false;
            for (j= 0; j < listaTemporalVector.GetLength(0) - 1; j++)
            {
                if (listaTemporalVector[j] > listaTemporalVector[j+1])
                {
                    temp= listaTemporalVector[j];
                    listaTemporalVector[j] = listaTemporalVector[j+1];
                    listaTemporalVector[j + 1]=temp;
                    algoCambiado = true;
                }
            }
            if (!algoCambiado)
            {
                parar = true;
            }
            i++;
        }

        listaTemporal.Clear();
        for (i = 0; i < listaTemporalVector.Count(); i++)//Pasamos de vector a
lista para adaptar al output de la función
        {
            listaTemporal.Add(listaTemporalVector[i]);
        }

        return (listaTemporal);
    }

    public double[] Percentil(List<double> lista, double percentil1, double
percentil2) //Devuelve el valor de la diferencia de la lista para 2 percentiles
    {
        double[] percentilesInput = new double[2];
        double[] percentilesOutput = new double[2];
        percentilesInput[0] = percentil1;
        percentilesInput[1] = percentil2;
        double condicion; //Dependiendo de si es entero o decimal el percentil
se calcula de una manera o de otra
        int indice;
        for (int i=0;i< percentilesInput.Length;i++)
        {
            condicion = lista.Count() * percentilesInput[i] / 100;
            if (condicion-Math.Floor(condicion)!=0)
            {
                indice = Convert.ToInt32(Math.Floor(condicion + 1)) - 1;
                if (indice < lista.Count())
                {
                    percentilesOutput[i] = lista[indice];
                }
                else percentilesOutput[i] = lista[lista.Count() - 1];
```

```
            }
            else
            {
                indice = Convert.ToInt32(condicion) - 1;
                if (indice<lista.Count()-1)
                {
                    percentilesOutput[i] = (lista[indice] + lista[indice + 1]) / 2;
                }
                else percentilesOutput[i] = lista[lista.Count() - 1];
            }
        }
        return (percentilesOutput);
    }

    public double Mean(List<double> lista)//Devuelve la media de una lista
    {
        double media;
        double total=0;
        for(int i=0; i<lista.Count();i++)
        {
            total = total + lista[i];
        }
        if (lista.Count() != 0)
        {
            media = total / lista.Count();
        }
        else media = 0;
        return (media);
    }

    public double StdDev(List<double> lista, double mean)//Devuelve
desviación estandar una lista
    {
        double std;
        double total = 0;
        for (int i = 0; i < lista.Count(); i++)
        {
            total = total + Math.Pow(lista[i] - mean, 2);
        }
        if (lista.Count() != 0)
        {
            std = Math.Sqrt(total / lista.Count());
        }
        else std = 0;
        return (std);
    }

    public void SetPosAccValues(int indice, List<double[]> diferencia, List<int>
zonas)//Ahorra repetir codigo. Con un indice de entrada y su correpondiente
lista + zonas establece el valor de Percentiles, media y STD
```

```
        {
            List<double> listaOrdenada = new List<double>();
            double percentil1 = 95;
            double percentil2 = 99;
            listaOrdenada = OrdenarDiferenciasPA(diferencia, zonas);
            if (listaOrdenada.Count() != 0)
            {
                PosAcc[indice, 0] = Percentil(listaOrdenada, percentil1, percentil2)[0];
                PosAcc[indice, 1] = Percentil(listaOrdenada, percentil1, percentil2)[1];
                PosAcc[indice, 5] = Mean(listaOrdenada);
                PosAcc[indice, 6] = StdDev(listaOrdenada, PosAcc[indice, 6]);
            }
        }

        public LectorMensaje AdaptarListaActual(LectorMensaje
listaActual)//Elimin de myList todos los Icao Add que no son del vehiculo
        {
            string icao = DGPSList.GetPlanI(0).GetICAOAdress();
            LectorMensaje lista = new LectorMensaje();
            for (int i = 0; i < listaActual.GetNumList(); i++)
            {
                if(listaActual.GetPlanI(i).GetICAOAdress()==
DGPSList.GetPlanI(0).GetICAOAdress())
                {
                    lista.AddPlanI(listaActual.GetPlanI(i));
                }
            }
            return (lista);
        }

        public double[] GetPosicion(int cont) //Miramos si los datos de posicion del
paquete vienen en WGS, del SMR, o en coordenadas cartesianas del MLAT
        {
            double[] posicion = new double[2];//Indice 0=X; indice 1=Y
            if ((myList.GetPlanI(cont).GetCartFromWGS84()[0] != Math.Pow(10, 8))
&& (myList.GetPlanI(cont).GetCartFromWGS84()[1] != Math.Pow(10, 8)))
            {
                posicion[0] = myList.GetPlanI(cont).GetCartFromWGS84()[0];
                posicion[1] = myList.GetPlanI(cont).GetCartFromWGS84()[1];
            }
            else if ((myList.GetPlanI(cont).GetMLATfromSMRX() != Math.Pow(10,
8)) && (myList.GetPlanI(cont).GetMLATfromSMRY() != Math.Pow(10, 8)))
            {
                posicion[0] = myList.GetPlanI(cont).GetMLATfromSMRX();
                posicion[1] = myList.GetPlanI(cont).GetMLATfromSMRY();
            }
            else if ((myList.GetPlanI(cont).GetCartX() != Math.Pow(10, 8)) &&
(myList.GetPlanI(cont).GetCartY() != Math.Pow(10, 8)))
            {
                posicion[0] = myList.GetPlanI(cont).GetCartX();
```

```
            posicion[1] = myList.GetPlanI(cont).GetCartY();
        }

        return (posicion);
    }

    public double[,] GetUpdateRate()
    {
        return this.updateRate;
    }
    public double[,] GetUpdatesUnitarios()
    {
        return this.updatesUnitarios;
    }
    public double[,] GetPositionAccuracy()
    {
        return this.PosAcc;
    }
    public double[,] GetProbOfMLATDetection()
    {
        return this.MLATDet;
    }
    public double[,] GetProbOfIdentification()
    {
        return this.Identification;
    }
    public double[,] GetProbOfFalseDetection()
    {
        return this.FalseDetection;
    }
    public double[,] GetProbOfFalseIdentification()
    {
        return this.FalseIdentification;
    }
    public List<double[]> GetDiferencias()
    {
        return this.diferencias;
    }
  }
}
```